

David Parsons · Kathryn MacCallum  
*Editors*

# Agile and Lean Concepts for Teaching and Learning

Bringing Methodologies from Industry  
to the Classroom

 Springer

# Agile and Lean Concepts for Teaching and Learning

David Parsons · Kathryn MacCallum  
Editors

# Agile and Lean Concepts for Teaching and Learning

Bringing Methodologies from Industry  
to the Classroom

 Springer

*Editors*

David Parsons  
The Mind Lab by Unitec  
Auckland, New Zealand

Kathryn MacCallum  
Eastern Institute of Technology  
Napier, New Zealand

ISBN 978-981-13-2750-6      ISBN 978-981-13-2751-3 (eBook)  
<https://doi.org/10.1007/978-981-13-2751-3>

Library of Congress Control Number: 2018955924

© Springer Nature Singapore Pte Ltd. 2019

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

The publisher, the authors and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, express or implied, with respect to the material contained herein or for any errors or omissions that may have been made. The publisher remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

This Springer imprint is published by the registered company Springer Nature Singapore Pte Ltd. The registered company address is: 152 Beach Road, #21-01/04 Gateway East, Singapore 189721, Singapore

# Foreword

The education sector has long had a learning problem. By that I mean it has often been slow to learn from ideas and processes that are adding real value in other sectors. Over the last few decades, lean and agile approaches have moved from the margins to the mainstream. The values, practices and techniques (and their hybrids) associated with these broad approaches have been adopted and adapted across many settings beyond where they were originally developed. Other than in education.

Parsons and MacCallum have done a great service to the education field by bringing together the emerging global thinkers and practitioner pioneers, who are translating and adapting agile and lean approaches to solve a broad range of important challenges in the education sector. Each of the chapters outlines experiments—both conceptual and practical—that offer rich learning for educators interested in how to achieve greater impact through the adoption of new values, beliefs, processes and techniques. This volume holds great promise in accelerating learning and further experimentation in the field in order to have a greater impact on outcomes for students.

As a former high school teacher and education academic, I was drawn to agile through sheer frustration. Continuous improvement efforts in education have a long history of being remarkably ineffective. I could see educational leaders and teachers being buried in thick, unrealistic improvement plans, never-ending lists of objectives and a myriad of milestones to be hit. Too often the improvement principles, processes and tools provided to educators were utterly disconnected from the real, messy and iterative teamwork needed to see lifts in teaching and learning outcomes. It became clear to me that new gains could emerge by providing educator teams with a *better how rather than another what*.

In 2012, I was fortunate in my work to be moving back and forth between education and the technology sector. I was eager to study how leaders and teams outside of education worked to solve complex problems in conditions of uncertainty. I studied innovation and improvement approaches from the health sector, leading technology companies in Silicon Valley, start-up entrepreneurs and design

organisations to understand the principles and methodologies they used to solve problems, create new solutions and drive continuous improvement.

As I met with and interviewed practitioners beyond education, I realised their team outcomes were often the result of shared values and repeatable processes. Agile and lean approaches seemed particularly relevant to the education sector. Teachers and students were often working collaboratively to solve problems, yet they struggled with: complexity and overwhelm; sustaining motivation over time; and the need to find approaches for effective collaboration and communication. I could see that ‘agile’ as an approach thrived when the problems were ambiguous or the situation was complex, and dealt with uncertainty by moving through rapid cycles of disciplined experimentation and continuous iteration and learning, rather than seeking to perfectly plan and then implement their way to success. A perfect match for educational contexts.

Brilliant educators around the world, many of whom I have been privileged to work with, have already been doing things that looked like the ‘agile’ and ‘lean’ processes I saw outside of education. The only difference was that they were doing them intuitively, without the consistency and discipline that comes from a clear, explicit and shared mindset, values and methodology. These educators were doing amazing things in their educational settings, but they often couldn’t put a name to how they were doing it, or share it as a process that their teams or students could easily repeat. I saw opportunities to make the work of teaching and educational change easier and more effective by adapting these ‘agile’ and ‘lean’ approaches to the specific challenge of improving student outcomes.

I launched [www.agileschools.com](http://www.agileschools.com) in January 2014 as a platform to share our methodologies and build the capabilities of educator teams to improve student learning. In partnership with teachers, we developed a process called Learning Sprints. Rather than seeking rapid large-scale system or massive organisational shifts, we are learning a simple yet powerful approach to creating an engine for educational improvement: fast-moving educator teams, working collaboratively on tough challenges, designing and testing solutions in context, and organically spreading and evolving practices to amplify impact.

As the chapters of this groundbreaking text show, innovative educators are applying agile and lean to a broad range of ‘sticky’ challenges in education and achieving promising results. I am increasingly convinced that educators and students can have a huge impact, even within the constraints of the established organisations and systems in which they do their work. It is time to empower educators and their students to start small, move fast and fail well, as they co-create brighter learning futures.

Towards better learning.

Sydney, Australia  
July 2018

Dr. Simon Breakspear  
Visiting Fellow, University of New South Wales  
Founder, Agile Schools

# Preface

## Introduction: The Motivation for This Book

As researchers and practitioners in both technology and education, we were motivated to put together this edited book by an interest in the potential links between innovative ideas developed in the software and manufacturing industries and opportunities for new thinking around the delivery of teaching and learning. We were aware that there was an increasing interest worldwide in how agile methods, developed in the software industry in the late 1990s, could be applied in all areas of education. Similarly, we have seen that lean thinking, developed from lean manufacturing in the Japanese car industry after the Second World War, has also become a major area of interest to educators. Since both agile and lean concepts have increasingly been integrated together in software development, it seemed appropriate to us to seek to draw together work from those who have been applying agile and/or lean techniques to educational practice.

It is evident that researchers and practitioners around the world have increasingly been exploring how agile and lean techniques might be used in educational contexts. For example, there is an established Lean Educator conference series, initially run in the U.S. and, more recently, in Europe, along with conferences for Lean in Higher Education, and there are many examples of lean thinking being applied in schools, while organisations such as Agile Schools and [scrum@school](mailto:scrum@school) have been promoting agile methods for teaching and learning. Other practitioners who are sharing ideas in this field include Scrum in School, and the group [agileineducation.org](http://agileineducation.org). These are just some examples of the many ways that educators have been embracing agile and lean ideas.

Despite this wealth of research and practice, we became aware that there was no existing publication providing a single point of reference for these ideas, particularly one that covered both agile and lean concepts which, although closely related together in the software industry, are less commonly linked in the educational context. Although a number of books and articles have been published in the area of lean thinking for education, much of this work has been applied at the management

level rather than focusing on teaching and learning. Further, some relevant conference outputs have unfortunately been poorly curated and some interesting contributions have therefore received less visibility than they deserved. For example, the proceedings of the early Lean Educator conferences in the U.S. are no longer available online. While much work has been published on the use of agile approaches to teach various aspects of software development, there is a relative lack of academic literature looking at how agile concepts can be applied more broadly in the classroom, despite the extensive work being undertaken by practitioners. We hope that publication of this book will increase recognition of the work that has already been done in this area of research and practice and, more importantly, inspire new and innovative applications of lean and agile concepts in education. Our own experience is that these ideas resonate strongly with educators, and there are many who would be interested in learning more about how to translate these innovations into the classroom. We hope that this book will inspire greater awareness of, and interest in, agile education and lean learning.

## **The Selection Process**

When we originally put out the call for proposals for this publication in early 2017, we were unsure what the level of response would be. In the event we were delighted at the number and breadth of abstracts that were submitted. After an initial review process, we invited selected authors to submit full chapters, which went through double-blind peer review, followed by an editorial review. From this rigorous selection process, 19 chapters have been chosen for publication. The international nature of our authors confirms that the idea of applying agile and/or lean concepts to education has a truly worldwide reach. Chapters have been contributed by authors from Australia, Denmark, Finland, Germany, Israel, Italy, Mauritius, Mexico, the Netherlands, New Zealand, Norway, Sweden, Switzerland, the U.K. and the U.S. These international contributors have helped us to provide a rich collection of global expertise for how agile and lean ideas can be applied in teaching and learning at all levels of education.

## **Book Structure**

We have organised the chapters in this book into a number of sections, beginning with chapters that provide various overviews of agile and lean approaches to education. We then include chapters that look at agile methods in the school classroom, followed by a section of chapters that address ways of reconceptualising learning environments using agile and lean approaches. The next section covers



agile and lean learning processes, and is followed by a section on using agile and lean methods to teach software development. We conclude with several chapters that report on agile and lean activities and games for the classroom.

## **Part I: Agile and Lean Concepts in Education**

The book opens with a section of overviews of agile and lean concepts in education, beginning with the chapter “[Agile Education, Lean Learning](#)”, written by the editors, which is intended to provide an introduction to agile and lean education as it has been explored so far in the literature. This chapter provides a broad and brief introduction to many aspects of this area and includes a summary diagram that draws together the major themes in the literature under the categories of values, principles and processes.

The second chapter is “[Agile Methodologies in Education: A Review](#)” by Filomena Ferrucci and Paolo Musmarra of the University of Salerno, Italy, and Pasquale Salza of USI Università della Svizzera italiana, Lugano, Switzerland. Like “[Agile Education, Lean Learning](#)”, this provides an overview of the literature, but in this case the focus is specifically on agile methods. The chapter includes details of the main characteristics of agile methods, focusing on two methods in particular (XP and Scrum), and looks at how these have been used in education. This chapter will be particularly informative for readers who are not already familiar with the details of the main agile methods.

The next chapter in this opening part is “[Practices of Agile Educational Environments: Analysis from the Perspective of the Public, Private, and Third Sectors](#)”, by Orit Hazzan (Technion—Israel Institute of Technology, Haifa, Israel) and Yael Dubinsky (Ness, Tel Aviv, Israel). The chapter provides an overview of agile principles in three different educational settings: academia, industry and the public sector. The authors build on their previous work in this area using two frameworks to explore the various ways that agile can go beyond conventional (formal or informal) educational systems to be applied to learning in other sectors and organisations.

The first part closes with “[Kaizen and Education](#)” by Peet Wiid of Kaizen Institute New Zealand. This chapter makes a distinction between the typical interpretations of Lean used in the western world and the concept and origins of kaizen. It provides an extensive overview of what kaizen means in practice and how it can be applied to education. While the five lean principles focus only on process, a kaizen culture in contrast is based on a broader set of seven principles: create customer value, eliminate waste, engage people, go to gemba, manage visually, process and results, and pull and flow. Such a kaizen culture can enable continuous improvement efforts in education.

## Part II: Agile Methods in the School Classroom

Part II of the book looks at agile methods in the school classroom, with three chapters that describe how schools in various countries have been implementing an agile approach. The first chapter in this part is “[Transforming Education with eduScrum](#)” by Willy Wijnands (Ashram College and eduScrum, Netherlands) and Alisa Stolze (eduScrum, Germany). In this chapter, the originator of eduScrum explains the motivation for creating this approach to classroom learning, outlines its core practices, processes and artefacts, and includes an external perspective on how eduScrum works based on a series of conversations with students, as well as two experience reports from students who have experienced eduScrum in practice.

In a similar vein, the chapter “[Getting Agile at School](#)” by Paul Magnuson, William Tihen, Nicola Cosgrove and Daniel Patton of the Leysin American School in Switzerland, explains how they began experimenting with Scrum as a way to structure self-regulated learning. They go on to describe how they use lean and agile techniques such as Kanban boards, sprints, burndown charts and retrospectives in class. Teaching and learning is based on action research cycles of planning, doing, reflecting and re-doing, built around ten practices of an agile mindset. The overall approach is to make small adjustments to multiple practices. The key message is that incremental change, shared by many, can be a powerful tool to create effective learning.

The third and final chapter that reports on the agile school classroom is “[Bringing the Benefits of Agile Techniques Inside the Classroom: A Practical Guide](#)” by Ilenia Fronza, Nabil El Ioini and Claus Pahl of the Free University of Bozen-Bolzano, Italy, and Luis Corral of Monterrey Institute of Technology and Higher Education, Queretaro, Mexico. Later in the book, there are several chapters that explore the use of agile and lean techniques to teach software development at the higher levels of education. In contrast, this chapter looks at ways in which software development can be taught to younger school students and examines how various agile techniques such as user stories and pair programming can help school students to become effective ‘end-user programmers’, learning both to write code and to understand software engineering concepts and practices.

## Part III: Reconceptualising Learning Environments Using Agile and Lean Approaches

The next part looks at how we might use agile and lean approaches to reconceptualise how we approach and manage learning environments. The chapters in this part explore different ways in which agile and lean thinking can help us to redesign significant aspects of the education process. In the chapter “[Lean and Agile Higher Education: Death to Grades, Courses, and Degree Programs?](#)”, Guttorm Sindre of the Norwegian University of Science and Technology discusses how lean and

agile education might be radically different from plan-based education. This chapter suggests that information technology could support a fine-grained matching of student learning outcomes with competencies needed by employers, and could enable agile study choices by the students themselves. One advantage of such an agile and lean approach to higher education provision would be that it would be better able to respond to the speed of technological progress.

The second chapter in this part is “[Leveraging Agile Methodology to Transform a University Learning and Teaching Unit](#)” by Madelaine-Marie Judd and Heidi Christina Blair of Griffith University, Australia. This chapter presents a case study of an Australian Learning and Teaching Unit in which an agile approach was used to improve quality across the institution. Recommendations and strategies for heightening buy-in and active staff engagement are provided, along with ways of giving clear and transparent communication to ensure widespread adoption. The agile approach led to unexpected gains in confidence and self-efficacy in the stakeholders. The authors also provide recommendations that may apply to a range of other industries and sectors.

This part concludes with the chapter “[Lean and Agile Assessment Workflows](#)” by Michael Striewe from the University of Duisburg-Essen, Germany. Assessment is a core component of education and is frequently the one that is least likely to embrace change. This chapter discusses a structured approach to assessment planning and organisation that is inspired by Kanban-style notations. Based on breaking down assessments into their essential elements and phases, it is a more agile approach to planning than traditional workflow and process models. The extensive analysis of all stages of the assessment process provides a model that maps the value stream clearly and assists in adaptation.

## **Part IV: Agile and Lean Learning Processes**

The next part includes chapters that look at agile and lean learning processes, specifically, agile goal setting, reflective practice and the management of risk. The first of these chapters is “[Criterion-Based Grading, Agile Goal Setting, and Course \(Un\)Completion Strategies](#)” by Essi Isohanni and Pietari Heino from Tampere University of Technology, Finland, and Petri Ihantola, and Tommi Mikkonen from the University of Helsinki, Finland. This chapter looks at agile ways of personalising the learning experience for large groups of students with heterogeneous backgrounds and different learning goals. The authors describe how each student sets a personal target grade and decides how much effort they are willing to invest in the course depending on their individual needs. To enable such a setup, course assignments are divided into different levels and the grading directs the students in choosing which assignments to work on to meet the goals they have set themselves. Furthermore, the students can change their target grade during the course in an agile manner.

The next chapter in this part is “[Teaching and Fostering Reflection in Software Engineering Project Courses](#)” by Håkan Burden (RISE Viktoria

and Chalmers/University of Gothenburg, Sweden) and Jan-Philipp Steghöfer (Chalmers/University of Gothenburg, Sweden). Agile processes such as sprint retrospectives in Scrum or learning feedback loops in XP make extensive use of reflection. Engineering education also emphasises the importance of reflective practice, e.g. in Kolb's learning cycle and Schön's reflection-in/on-action. This chapter provides a toolkit for reflective practice for students to reflect on the application of a software process and also to reflect on their learning process. A cognitive apprenticeship approach is used, with both teachers and students going through reflective processes with the aim of continuous improvement.

The last chapter in this part is "[Lean Learning of Risks in Students' Agile Teams](#)" by Chaitra Thota, Wentao Wang, Xiaoyu Jin, Nan Niu and Carla C. Purdy of the University of Cincinnati, USA. The chapter addresses the idea that risk is an essential part of software development and explores how students in agile software teams perceive, prioritise and mitigate risk over multiple development cycles. Reporting on a study of undergraduate students in agile teams, the authors show that the students' risk management strategies were both collaborative and lean. For example, they used social media features like collaborative bookmarking and tutorial co-creation to address technology-centric risks and reduced wasted effort on non-actionable risks. The chapter describes how linking collaboration and waste elimination provided additional insights into teaching a wider range of lean principles in agile settings.

## **Part V: Using Agile and Lean Methods to Teach Software Development**

As indicated in the opening chapters of this book, the most common way in which agile and lean concepts have been applied in education has been to teach these concepts to students who are learning to develop software. Therefore, it is no surprise that some chapters of this type have been included in this book, in this part on using agile and lean methods to teach software development. The first of these is the chapter "[Applying Lean Learning to Software Engineering Education](#)" by Robert Chatley of Imperial College London, UK. This chapter describes the teaching of lean and agile techniques in a software engineering programme that is centred on the tools, techniques and issues that feature in the everyday life of a professional software developer working in a modern team. The author notes that aligning teaching methods with the principles of lean software delivery sustains high-quality learning experiences. A modified lecture-based course with tight feedback loops is compared with a project-based course where students put agile methods into practice themselves, working in teams to build a substantial software system over a number of months.

The second chapter in this part is "[Developing a Spiral Curriculum for Teaching Agile at the National Software Academy](#)" by James Osborne, Carl Jones and

Wendy Ivins from Cardiff University, UK. This chapter provides an experience report from the National Software Academy (NSA), a centre of excellence for Applied Software Engineering that works in partnership with the Welsh Government and industry leaders. The authors describe their innovative, industry-focused qualifications which use agile methods to facilitate project-based learning using projects provided by industry partners. The chapter outlines how a spiral curriculum has been developed for teaching agile methods that progressively introduces complexity whilst building on previous learning.

The third and final chapter in this part is “[Agile Approaches for Teaching and Learning Software Architecture Design Processes and Methods](#)” by Muhammad Aufeef Chauhan (Netcompany A/S, Copenhagen, Denmark) Christian W. Probst from Unitec Institute of Technology, New Zealand and M. Ali Babar, University of Adelaide, Australia. In this chapter, the authors provide agile teaching and learning approaches for software architecture analysis, design and evaluation. The chapter identifies key characteristics of agile software architecture processes and the roles of agile teams in software architecture. Case studies are presented from agile teaching and learning approaches over 2 years of software architecture courses, specifically focusing on learning activities that can support lean education and collaboration among the students and course instructors.

## **Part VI: Agile and Lean Activities and Games for the Classroom**

The final part of the book comprises three chapters that look at agile and lean activities and games for the classroom. The first of these is “[A Practical Approach to Teaching Agile Methodologies and Principles at Tertiary Level Using Student-Centred Activities](#)” by Visham Hurbungs and Soulakshmee Devi Nagowah of the University of Mauritius. The chapter describes how university students can be made familiar with agile practices currently used by the software industry using game-like active learning. In this chapter, the main focus is how team-based activities and student-centred group work have helped university students learn, understand and apply agile concepts such as Scrum, user stories, Extreme Programming (XP), Lean, Kanban and Test-Driven Development (TDD) through a series of learning games.

Another chapter that has a similar focus is “[Using Agile Games to Invigorate Agile and Lean Software Development Learning in Classrooms](#)” by Rashina Hoda of the University of Auckland, New Zealand. The chapter reports on the use of four agile games for learning fundamental agile and lean concepts such as iterative and incremental delivery, collaborative estimation, pair programming and work-in-progress limits. The author explains how agile games can support effective learning, learner engagement and team building, while warning that effective facilitation and debriefing sessions are essential to the success of using these games in the classroom.

The final chapter in the book is “[Red-Green-Go! A Self-Organising Game for Teaching Test-Driven Development](#)” by Suzanne M. Embury, Martin Borizanov and Caroline Jay from the University of Manchester, UK. Unlike the two previous chapters, this one does not explore the application of pre-written games. Rather, it describes the development of a new game-like classroom activity called Red-Green-Go!, a board game for learning Test-Driven Development (TDD) that allows students to tailor the learning experience to suit their own level of experience and skill. The game makes use of self-organising teams, big visible charts, frequent feedback and reflection to create a self-paced teaching activity. It guides pairs of students through the TDD cycle, as well as introducing students to different pair-coding styles.

We hope that you will find this book a valuable source of ideas about how agile and lean concepts can be applied to teaching and learning. Successfully bringing methodologies from industry to the classroom is a challenge for educators, since it requires a deep understanding of which aspects of these methods can provide value to the learner largely unchanged, and which may need to be extensively reinterpreted to ensure that they work within the classroom context. There is no doubt that many educators have found agile and lean thinking a powerful new way to approach their profession practice, and we are pleased to have been able to capture some of their wisdom in this book. We very much hope that this publication will inspire other researchers and practitioners to embrace change and further develop this innovative and powerful new approach to teaching and learning.

Auckland, New Zealand  
Napier, New Zealand

David Parsons  
Kathryn MacCallum

# Acknowledgements

The editors would like to acknowledge the support and encouragement of Nick Melchior, Executive Editor (Education) at Springer. Nick originally approached us in October 2016 at the 15th World Conference on Mobile and Contextual Learning, Sydney, Australia, where we first pitched the idea of an edited book on agile and lean concepts in education. We were very pleased that Nick readily took our proposal on board and has helped us through all the stages of bringing this book from concept to publication.

## International Review Board

Ensuring the quality of a volume such as this relies heavily on having a credible and committed international review board. The editors would like to acknowledge the following members of the board who gave their time and expertise to the double-blind peer-review process.

Vernon Bachor  
Heidi Blair  
Sofia Sousa Brito  
Håkan Burden  
Robert Chatley  
Aufeef Chauhan  
Dawit Demissie  
Melinda Dixon  
Andreas Drechsler  
Suzanne M. Embury  
Ilenia Fronza  
Tim Gander  
Orit Hazzan  
Rashina Hoda

Visham Hurbungs  
Wendy Ivins  
Mary Jacob  
Carl Jones  
Madelaine-Marie Judd  
Tanya Linden  
Lucie Lindsay  
Deborah J. McCraw  
Paul Magnuson  
Vic Matta  
Tommi Mikkonen  
Soulakshmee D. Nagowah  
Nan Niu  
James Osborne  
Samiaji Sarosa  
Jason Sharp  
Guttorm Sindre  
Jan-Philipp Steghöfer  
Michael Striewe  
Allan Sylvester  
Bill Tihen  
John Woollard



# Contents

## **Part I Agile and Lean Concepts in Education**

<b>Agile Education, Lean Learning</b> . . . . .	3
David Parsons and Kathryn MacCallum	
<b>Agile Methodologies in Education: A Review</b> . . . . .	25
Pasquale Salza, Paolo Musmarra and Filomena Ferrucci	
<b>Practices of Agile Educational Environments: Analysis from the Perspective of the Public, Private, and Third Sectors</b> . . . . .	47
Orit Hazzan and Yael Dubinsky	
<b>Kaizen and Education</b> . . . . .	63
Peet Wiid	

## **Part II Agile Methods in the School Classroom**

<b>Transforming Education with eduScrum</b> . . . . .	95
Willy Wijnands and Alisa Stolze	
<b>Getting Agile at School</b> . . . . .	115
Paul Magnuson, William Tihen, Nicola Cosgrove and Daniel Patton	
<b>Bringing the Benefits of Agile Techniques Inside the Classroom: A Practical Guide</b> . . . . .	133
Ilenia Fronza, Nabil El Ioini, Claus Pahl and Luis Corral	

## **Part III Reconceptualising Learning Environments Using Agile and Lean Approaches**

<b>Lean and Agile Higher Education: Death to Grades, Courses, and Degree Programs?</b> . . . . .	155
Guttorm Sindre	

**Leveraging Agile Methodology to Transform a University Learning and Teaching Unit** ..... 171  
Madelaine-Marie Judd and Heidi Christina Blair

**Lean and Agile Assessment Workflows** ..... 187  
Michael Striewe

**Part IV Agile and Lean Learning Processes**

**Criterion-Based Grading, Agile Goal Setting, and Course (Un)Completion Strategies** ..... 207  
Petri Ihantola, Essi Isohanni, Pietari Heino and Tommi Mikkonen

**Teaching and Fostering Reflection in Software Engineering Project Courses** ..... 231  
Håkan Burden and Jan-Philipp Steghöfer

**Lean Learning of Risks in Students’ Agile Teams** ..... 263  
Wentao Wang, Chaitra Thota, Xiaoyu Jin, Nan Niu and Carla C. Purdy

**Part V Using Agile and Lean Methods to Teach Software Development**

**Applying Lean Learning to Software Engineering Education** ..... 285  
Robert Chatley

**Developing a Spiral Curriculum for Teaching Agile at the National Software Academy** ..... 303  
James Osborne, Wendy Ivins and Carl Jones

**Agile Approaches for Teaching and Learning Software Architecture Design Processes and Methods** ..... 325  
Muhammad Afeef Chauhan, Christian W. Probst and Muhammad Ali Babar

**Part VI Agile and Lean Activities and Games for the Classroom**

**A Practical Approach to Teaching Agile Methodologies and Principles at Tertiary Level Using Student-Centred Activities** ..... 355  
Visham Hurbungs and Soulakshmee Devi Nagowah

**Using Agile Games to Invigorate Agile and Lean Software Development Learning in Classrooms** ..... 391  
Rashina Hoda

**Red-Green-Go! A Self-Organising Game for Teaching Test-Driven Development** ..... 415  
Suzanne M. Embury, Martin Borizanov and Caroline Jay

# Contributors

**Heidi Christina Blair** Griffith University, Brisbane, Australia

**Martin Borizanov** School of Computer Science, The University of Manchester, Manchester, UK

**Håkan Burden** RISE Viktoria, Gothenburg, Sweden; Chalmers | University of Gothenburg, Gothenburg, Sweden

**Robert Chatley** Department of Computing, Imperial College London, London, UK

**Muhammad Ali Babar** The University of Adelaide, Adelaide, Australia

**Muhammad Aueef Chauhan** Netcompany A/S, Copenhagen, Denmark

**Luis Corral** Monterrey Institute of Technology and Higher Education, Queretaro, Mexico

**Nicola Cosgrove** Leysin American School, Leysin, Switzerland

**Yael Dubinsky** Ness Israel, Tel Aviv, Israel

**Nabil El Ioini** Free University of Bozen-Bolzano, Bolzano, Italy

**Suzanne M. Embury** School of Computer Science, The University of Manchester, Manchester, UK

**Filomena Ferrucci** University of Salerno, Fisciano, Italy

**Ilenia Fronza** Free University of Bozen-Bolzano, Bolzano, Italy

**Orit Hazzan** Technion – Israel Institute of Technology, Haifa, Israel

**Pietari Heino** Tampere University of Technology, Tampere, Finland

**Rashina Hoda** SEPTA Research, Department of Electrical and Computer Engineering, The University of Auckland, Auckland, New Zealand

**Visham Hurbungs** Department of Software and Information Systems, Faculty of Information, Communication & Digital Technologies, University of Mauritius, Reduit, Mauritius

**Petri Ihantola** University of Helsinki, Helsinki, Finland

**Essi Isohanni** Tampere University of Technology, Tampere, Finland

**Wendy Ivins** Cardiff University, Cardiff, UK

**Caroline Jay** School of Computer Science, The University of Manchester, Manchester, UK

**Xiaoyu Jin** Department of EECS, University of Cincinnati, Cincinnati, OH, USA

**Carl Jones** Cardiff University, Cardiff, UK

**Madelaine-Marie Judd** The University of Queensland, Brisbane, Australia

**Kathryn MacCallum** Eastern Institute of Technology, Napier, New Zealand

**Paul Magnuson** Leysin American School, Leysin, Switzerland

**Tommi Mikkonen** University of Helsinki, Helsinki, Finland

**Paolo Musmarra** University of Salerno, Fisciano, Italy

**Soulakshmee Devi Nagowah** Department of Software and Information Systems, Faculty of Information, Communication & Digital Technologies, University of Mauritius, Reduit, Mauritius

**Nan Niu** Department of EECS, University of Cincinnati, Cincinnati, OH, USA

**James Osborne** Cardiff University, Cardiff, UK

**Claus Pahl** Free University of Bozen-Bolzano, Bolzano, Italy

**David Parsons** The Mind Lab by Unitec, Auckland, New Zealand

**Daniel Patton** Leysin American School, Leysin, Switzerland

**Christian W. Probst** Unitec Institute of Technology, Auckland, New Zealand

**Carla C. Purdy** Department of EECS, University of Cincinnati, Cincinnati, OH, USA

**Pasquale Salza** USI Università della Svizzera italiana, Lugano, Switzerland

**Guttorm Sindre** Department of Computer Science, Norwegian University of Science and Technology (NTNU), Trondheim, Norway

**Jan-Philipp Steghöfer** Chalmers | University of Gothenburg, Gothenburg, Sweden

**Alisa Stolze** eduScrum, Berlin, Germany

**Michael Striewe** University of Duisburg-Essen, Essen, Germany

**Chaitra Thota** Department of EECS, University of Cincinnati, Cincinnati, OH, USA

**William Tihen** Garaio, Bern, Switzerland

**Wentao Wang** Department of EECS, University of Cincinnati, Cincinnati, OH, USA

**Peet Wiid** Kaizen Institute, Auckland, New Zealand; Manukau Institute of Technology, Auckland, New Zealand

**Willy Wijnands** eduScrum, Alphen aan den Rijn, The Netherlands

**Part I**  
**Agile and Lean Concepts in Education**

# Agile Education, Lean Learning



David Parsons and Kathryn MacCallum

**Abstract** There is growing interest in applying both agile and lean concepts in the classroom to improve educational experiences. In this chapter, we draw together the disparate ideas of these two fields from industrial practice and the existing work within this area to develop and frame the major concepts of agile and lean thinking for teaching and learning. The chapter summarises the key ideas relating to how values, processes and techniques from agile software development, overlaid with related concepts from lean thinking, can be translated to the broader needs of education across disciplines for students of all ages. From a review of the available literature, we draw out a simple conceptual framework that we use to present the key themes from the literature around how both agile and lean approaches can be used in education. We conclude by providing some insights into how agile and lean teaching and learning can be applied as an integrated set of educational concepts by identifying the essential skills and practices that can be transferred to the classroom.

**Keywords** Agile · Lean · Schools · Education · Teaching · Learning

## 1 Introduction

In this chapter, we look at how agile and lean ideas have been taken from industry and applied in the classroom. Then we take the key themes and concepts from both these approaches and integrate them into a simple conceptual framework that identifies the main ways in which these two industrial practices have been applied to teaching and learning, regardless of subject discipline and level of education. In doing so, we aim to draw out the important skills and practices that are required in making this transition from one domain to another.

---

D. Parsons (✉)  
The Mind Lab by Unitec, Auckland, New Zealand  
e-mail: [daveparsonsnz@gmail.com](mailto:daveparsonsnz@gmail.com)

K. MacCallum  
Eastern Institute of Technology, Napier, New Zealand

© Springer Nature Singapore Pte Ltd. 2019  
D. Parsons and K. MacCallum (eds.), *Agile and Lean Concepts for Teaching and Learning*, [https://doi.org/10.1007/978-981-13-2751-3\\_1](https://doi.org/10.1007/978-981-13-2751-3_1)

## ***1.1 Using Agile to Teach Agile and Lean to Teach Lean***

The bulk of the literature around applying agile methods to teaching and learning focuses on various aspects of the design and development of software systems. These studies typically focus on the application of agile methods for teaching software engineering (e.g. Melnik & Maurer, 2003), redesigning teaching approaches (e.g. Layman, Cornwell, & Williams, 2006), supporting teamwork (e.g. Rico & Sayani, 2009) and applying agile approaches (like Scrum) to support student development (e.g. Mahnic, 2012). Much of this effort is justified on the grounds that it prepares students studying software systems development with the required industry skills for entering the profession (e.g. Bruegge, Reiss, & Schiller, 2009), rather than providing any exploration of whether agile methods themselves offer new and valuable approaches to pedagogy across a range of educational contexts. Although some authors have attempted to broaden their focus on education, such as Hazzan and Dubinsky's (2014) references to the Finnish education system, many still mainly address software engineering issues.

A similar limitation applies to much of the literature around lean approaches to education. The focus of much work in this area is narrowly directed to higher education institutions, either related to their administrative processes or the use of lean principles to teach certain technical subjects. Most of the chapters in Alves, Flumerfelt, and Kahlen (2016), for example, describe lean approaches to the teaching of engineering or related subjects, typically in higher education, or lean processes at the institutional level. The idea that lean is a mechanism for streamlining the administrative processes of educational institutions is examined even more explicitly in Balzer (2010), where teaching and learning is excluded from the analysis entirely. Francis (2014) notes that much of the literature about applying lean to education reports on institutions adopting the industrial model uncritically, and questions whether this is appropriate. This question is further reinforced by Comm and Mathaisel (2005) who observe that many educational institutions regard lean as a process of cost cutting and outsourcing, rather than one that aims to meet the needs of the learner.

Overall, in the literature we see a lot of work describing the use of agile to teach agile, and lean to teach lean, or work that applies industrial disciplines to the processes of education, as if teaching and learning is the same kind of product as cars or software. Less evident than these approaches, though perhaps more important to education as a whole, is literature that addresses how agile and lean methods can be used more broadly within the classroom, beyond the systems development, engineering or management contexts. That is the focus of this chapter.

## ***1.2 Agile and Lean Education***

Although there is a significant body of work that addresses either agile or lean concepts applied to education, they are rarely dealt with together in this context. In



this chapter, however, we seek to identify complementary themes and ideas from both strands of research. The main motivation for this is that both agile and lean approaches have been successfully integrated into software development, a combination that has been extensively discussed in the literature for that field of practice and related areas such as supply chain (Naylor, Naim, & Berry, 1999). One popular example would be the ‘Scrumban’ software development method, which is a combination of the agile Scrum process and the lean Kanban technique (Ladas, 2008). The two approaches are seen as partly overlapping and complementary, for example Petersen (2011) notes that both share the same goals and define similar principles, though agile has more of a process focus.

In the following sections, we first examine agile and lean approaches in isolation, then bring them together to explore their complementary philosophies. We begin by providing a brief overview of agility and agile software development methods before moving on to how agile methods can be applied more generally in the classroom. We then explore in more detail how agile values, processes and techniques may be reinterpreted in the context of teaching and learning. This is followed by coverage of lean concepts, and how they relate to education, again focusing on values, processes and techniques. The final section of the chapter draws together these various themes and ideas and provides an analysis of how agile and lean can effectively transition from their industrial roots to teaching and learning.

## 2 Agile Methods

Agile methods for software development evolved as a response to the changing dynamics of the software industry in the 1990s. The focus on customer needs and being responsive to changes in requirements at any phase of development were seen as increasingly important (Dingsøy, Nerur, Balijepally, & Moe, 2012). The broader concept of agility emerges from the four values and 12 principles expressed within the Manifesto for Agile Software Development (Agile Alliance, 2001). Although agility itself is not formally defined within the Manifesto (its principles only refer occasionally to ‘agile processes’), the concept of welcoming change is embodied within these principles, along with collaboration, motivation and reflection. Agility within software development is underpinned by these concepts to rapidly create, embrace and learn from change while contributing to perceived customer value (Conboy, 2009).

### 2.1 Agile in Education

Embracing change is a fundamental principle of agile software development (Beck & Andres, 2005). In a rapidly changing world, education also has a pressing need to do the same. The aspects of change, flexibility and leanness (a concept borrowed from lean management) introduced above are just as important in education as for

software development. Educators are constantly subjected to change as new and emerging techniques, tools and ways of teaching are assessed and implemented to support the dynamic needs of today's learners (Hew & Brush, 2007).

It is notable that much of the material available on the specifics of agile teaching and learning comes not from academic sources but from other individuals and organisations who have a practitioner focus. Briggs (2014) notes that agile approaches are being tried out in schools around the world, citing India and Brazil as two examples. Other high-profile practitioners are active in the United States (Agile Classrooms), Peru (Laboratoria), Australia (Agile Schools), and The Netherlands (eduScrum and Scrum@School) among others. Unfortunately, there is a lack of formal literature about these initiatives, but a number of academics have also addressed the relationships between agile and education. For example, Stewart, DeCusatis, Kidder, Massi, and Anne (2009) assert that software development and education have similar methodologies. Both require detailed planning and scheduling, rely on constant assessment and feedback from all involved, and have stringent quality and scheduling criteria. Exploring this link further, they summarise that agile methods can be incorporated into the learning context to enhance project-based learning, collaborative experiences and student-led learning, and can support learning that is goal driven rather than plan-driven.

## ***2.2 Mapping Agile Methods to Classroom Practice***

Problems arise, however, when trying to identify more specific mappings between agile methods and classroom practice. Many of the examples in the literature outline agile education in very broad terms. For example, Dewi and Muniandy (2014) draw together some literature on agile approaches to teaching and learning, but their summary of techniques is very general (e.g. small group discussion, problem-based learning, blended learning, cooperative learning, etc.) and it is hard to link these practices specifically to any agile sources. Obrist et al. (2011) examine how the adoption of agile team roles (such as testers, informants and design partners) within design-based activities can clarify and strengthen team participation. However, this discussion only focuses on one aspect of agile, and again the mapping between the specifics of agile industry practice and what might be termed agile teaching practice is at a somewhat abstract level.

Some authors have tried to more closely tie agile methodologies to agile teaching. In particular, Meerbaum-Salant and Hazzan (2010) distil agile practices into three aspects; a pedagogical class management aspect, a social aspect and a project management aspect. In their Agile Constructionist Mentoring Methodology (ACMM) they define a teaching process that supports teachers in guiding their students in software projects. This methodology is based on the seven categories of Shulman's Teacher Knowledge Base Model and constructionism. Their approach is presented within the context of teaching software development, but also has application to other classroom environments. However, these aspects are still largely conceptual in

nature, and although they try to connect agile methodology to teaching and learning this is still very much a broad-brush approach.

These examples perhaps indicate that interpretations of what is ‘agile’ in the classroom need to be more clearly defined. In addition, more concrete examples are needed to illustrate the specific ways that educators can use agile approaches to transform teaching and learning.

### 3 Reinterpreting Agile Practice for Teaching and Learning

If we are to be able to make specific recommendations about how educators can use agile practices in the classroom, then we have to break them down into implementable strategies. To do this, we analyse agile teaching and learning at the level of values, processes and techniques.

#### 3.1 *Agile Values*

Four agile values are expressed in the Manifesto for Agile Software Development (Agile Alliance, 2001). Peha (2011) re-envisions how the four values of the manifesto apply to education in ‘The Agile Schools Manifesto’. His version states that:

We are uncovering better ways of educating children by doing it and helping others do it.  
Through this work we have come to value:  
Individuals and interactions over processes and tools;  
Meaningful learning over the measurement of learning;  
Stakeholder collaboration over constant negotiation;  
Responding to change over following a plan.

It should be noted that some earlier authors have made similar attempts to rewrite the manifesto for agile software development with educational contexts in mind. For example, an alternative reworking of the manifesto’s four values was offered by Stewart et al. (2009). However, this interpretation focused particularly on active and collaborative learning. Another reworking by Tesar and Sieber (2010) suggested a similarly narrow focus on ‘agile e-learning’. Such efforts tend to drift away somewhat from the original motivations of the values of the manifesto. Kamat (2012) changed every component of the values, while Krehbiel et al. (2017) used a longer and entirely different list of values, which seems to negate the idea that the existing values can be reinterpreted for education, rather than replaced. Given the limitations of these proposals, we will focus here on Peha’s version of the manifesto.

There are two questions to be considered when looking at the reinterpretation of the agile manifesto in the context of education. In particular, we need to look at what remains the same, what has been changed, and why? In addressing these questions,

we can explore how agile methods can be seen as being directly applicable to teaching and learning, and also to identify the key measures of progress in education, and the main stakeholders.

What remains the same in Peha's agile values are lines 1 and 4. The fact that these are unchanged is perhaps at the heart of Peha's approach, which is that agile thinking can be applied to education with many of its fundamental ideas intact. However, some changes are essential to align the manifesto to education, in particular the replacement of 'working software' in line 2 as being the primary measure of progress with 'meaningful learning'. Another fundamental change is the replacement of 'customer' with 'stakeholder' in line 3. The concept of 'stakeholder' in education is very important, since it would not be sufficient to simply replace 'customer' with 'student'. Stakeholders in education are the students, but also their caregivers, the learner's wider family, teaching and administration staff, school boards, local and national education authorities and a range of other interested parties. Effective change within educational needs to engage with the beliefs, values, vision and needs of all stakeholders (Zion, 2009). On the right-hand side of the values, the replacement of 'contract negotiation' with 'constant negotiation' is interesting, since many educators feel that they are constantly dealing with changing policies and procedures rather than being able to focus on teaching and learning. The final change Peha makes is to replace 'comprehensive documentation' in line 2 with 'measurement of learning'. Again, many educators feel that assessment takes precedence over learning in many jurisdictions.

In addition to the four values, the Agile Manifesto also includes 12 principles. These can be seen as a set of competencies of individuals and teams that together enable agile development. In reference to education, Kropp, Meier, Mateescu, and Zahn (2014) point out that these agile competencies are not limited to technical skills, but also encompass social skills. While technical skills are important, the agile values and principles are much broader than these alone. Kropp et al. (2014) stress both agile values and attitudes, and outline an agile model based on learning through personal experience, social learning and learning through realistic discourse, and construction of values and value identity. Collaboration and communication are seen as key to these ideals.

It should be noted that the agile values expressed in the literature are generally taken to be those outlined in the Manifesto, but there are other definitions. For example, Beck and Andres (2005) defined the five values of eXtreme Programming (XP, an agile software development method) as communication, simplicity, feedback, courage and respect, which Meerbaum-Salant and Hazzan (2010) use as the basis for the ACMM.

Another approach to capturing the essence of agile in education is the Agile Compass (Delhij et al., 2016). Like the agile manifesto, the compass encapsulates a journey from one state of practice to another, in this case from prescriptive to iterative, content to culture, evaluation to visible feedback and reflection, control to trust and competition to collaboration. Although this interpretation is not based directly on the concepts of the original agile manifesto, it does capture the key idea of how agile transformation leads to change across a range of learning areas. Another

set of agile ‘advantages’ are outlined by Scrum@School (n.d.) and no doubt there are other examples. The key issue in interpreting these sets of values is to recognise what practical changes they imply in the classroom. In this chapter we discuss a range of examples from the literature where agile values have been put into practice.

### 3.2 *Agile Processes*

Educational institutions are full of processes, and some of these might benefit from a more agile, adaptive, change-embracing approach. Agile processes are essentially iterative cycles of creation and reflection, where budgets and timescales are fixed, and quality is a given. Thus, agile planning is based on trading off coverage against priority. Agile processes tend to either emphasise engineering (e.g. XP) or management (e.g. Crystal). Another essential feature of agile methods is that they are iterative and adaptive. An example of this approach in an educational context is the Successive Approximation Model (SAM) for developing learning products, with an emphasis on iterative, short work cycles (Allen, 2012).

Agile processes are not just adaptive. Being adaptive is often emphasised when discussing agile organisations, but adaptivity alone would lead to lack of direction and strategy. Agile organisations and teams need to be adaptive within a controlled, managed and interactive framework, so that in adapting to change, they are able to realistically negotiate priorities and resources with all stakeholders. It is this framework for adaptivity that agile processes provide.

Scrum is a very popular agile software development process with a management focus, which Peha (2011) suggests would help to establish shared practices supported by clear ownership and roles across the school. In a Scrum process, the product backlog (of user stories, which capture user requirements) is broken down into a series of sprints (timeboxed activities). In agile methods such as Scrum, the user stories are often captured using index cards on storyboards, which let people see what others are doing and help management track progress and plan (Cohen, Lindvall, & Costa, 2004). In each sprint, a priority list of stories (the sprint backlog) is chosen for completion, during which time the stories will move across the storyboard as they are completed. The sprint lasts for a certain period of time (e.g. 2 weeks, 30 days etc.). There are daily stand-up meetings during the sprint, and at the end of each sprint a working increment of the software is delivered. In other words, it is only a successful sprint if it delivers something useful. Peha (2011) re-applies the sprint concept to schools, where they can incorporate learning backlogs, rapid turnaround of learning and integrated assessment. He states that breaking down the traditional teaching durations of months or the school year to short sprints would help make learning more focused and reflective. This would mean that time is not wasted on ill-conceived ideas over long periods of time but that students can be reflective and able to adapt, with learning constantly reassessed and reshaped as learners progress through each sprint. This, implemented alongside backlogs of learner stories, would help to track learning progress and identify and focus learning goals.

### 3.3 Agile Techniques

Although a reworking of the agile values and principles provides a set of relevant competencies for agile teaching and learning, and processes provide overall frameworks for activities, it is at the level of techniques that we can really identify specific classroom practices that can be considered agile. Fortunately, there is a broad range of agile techniques that can easily be adapted to the classroom.

A number of authors have started to identify specific techniques from agile software development that can be applied to teaching and learning. For example, Peha (2011) examines a number of agile techniques (which he calls patterns of practice) that can be applied in schools, these include stand-up meetings, paired teaching, user stories and test-first development. Stand-up meetings could be used for both staff meetings and classroom meetings with students, though as Peha notes this would require certain levels of autonomy for both teachers and students to be effective. Paired teaching can help the sharing of knowledge and expertise and improve learning as teachers can each lead different parts of the learning experience based on their particular expertise. Peha confines his discussion to only paired teaching, but the same idea could also apply to paired learning, whereby learners are paired up to support and facilitate learning. Switching roles within pairs between ‘driver’ and ‘navigator’, and regularly switching pair partners, are effective ways of sharing knowledge between peers. Further suggestions by Peha to adopt user stories and test-first development relate strongly to making learning more connected to the learners and responsive to individual learning. For example, user stories enable a teacher to re-state generic learning outcomes in terms of specific user stories where the students are considered as the ‘users’, making the learning standards more relevant to individuals. Similarly, test-first development would help to clarify learning targets and make learning achievements more visible and responsive to the needs of the learner.

In addition to those mentioned above, a range of agile techniques applied to education have been suggested in several other sources. In their review of the literature around the use of agile principles in active and cooperative learning, Stewart et al. (2009) report on teachers using stand-up meetings, retrospectives, rapid feedback, regular measures of progress and collaborative teams in their classes. Manamendra, Manathunga, Perera, and Kodagoda (2013) discuss how they used stand-up meetings to manage communication between research students and their supervisors. Kessler and Dykman (2007) also recommend stand-up meetings, and pairing, along with several other aspects of the Crystal Clear agile method including frequent delivery, reflection, improvement, osmotic communication and burn charts (information radiators). Allen (2012) stresses prototyping, which has specific roles in agile methods, both as initial throwaway prototypes (‘spikes’) and also as architectural prototypes that can be further evolved.

As indicated above, although Peha (2011) suggests paired teaching, the concepts behind the agile technique of pair programming can also be an effective approach to learning, as other authors have proposed. If we take away the task of programming, the other components of pairing remain valid, in particular the continuous inspection

and the ability of the navigator to think more broadly and strategically about the problem being solved than the driver. Such pairing can be seen in other contexts, for example Vanhoenacker (2015) points out that when a pilot and co-pilot are in the cockpit of an aircraft, regardless of which one of them has their hands on the controls at a particular time, they are both simultaneously flying the plane. It is a paired activity. Therefore, this idea of mutual support and peer learning can be applied in multiple contexts and with many kinds of participants from various stakeholder groups.

In addition to the techniques suggested in these studies, there are possibly other agile techniques that could be considered as relevant to teaching and learning. Some of those not mentioned explicitly above include refactoring, regression testing, colocation, common coding guidelines, continuous integration and single sourcing information (Parsons, Ryu, & Lal, 2007). Even techniques that seem very much rooted in the development of code, such as refactoring and continuous integration, can perhaps be reformulated for an educational setting. The important thing is to identify the transferable concepts behind such techniques.

## **4 Making Learning Agile**

Bringing these various ideas together, agile education might focus on the ideas of an iterative, adaptive process of student directed learning, built around learning stories created by the students themselves. Students would work mostly in pairs and self-organising teams, providing each other with constant support, feedback and mutual learning. Regular learning checks would take place through stand-ups and retrospectives, giving an opportunity for reflection on learning and embracing change where necessary. Students would be encouraged to develop broad skill sets and base their learning on real-world problems. Educators would act primarily as coaches, guiding learners rather than directing them, and the constant emphasis would be on meaningful learning above all other concerns.

## **5 Lean Manufacturing**

So far in this chapter we have been focusing on agile concepts and ideas and how they might be applied to education. We now turn our attention to the concepts and ideas of lean manufacturing, and how these have been transferred into the world of software development and, more importantly, on into the world of teaching and learning. So, what is lean, and how can it act as a complement to the agile ideas that we have been exploring so far in this chapter?

## ***5.1 From the Toyota Production System to Lean Software Development***

Lean is an approach within manufacturing, that has its roots in the Toyota Production System from the second half of the twentieth century. Lean production was conceptualised as a way to reduce waste, upskill workers, improve quality and provide more variety in products than was possible with mass production (Womack, Jones, & Roos, 1990). More recently, it has been applied to the development of software in what might be termed the post-agile period (Poppendieck & Poppendieck, 2003; Anderson, 2010). Since then, many people have been looking for ways of applying these lessons to their own work contexts by applying lean thinking, which addresses how the lean production ideas from the car industry can be applied to a range of other industries (Womack & Jones, 2003).

## ***5.2 Lean Concepts in Education***

An important question for educators is why they would concern themselves with concepts about reducing inventory and shortening the supply chain, if their product is something very different, like intellectual property and graduating students, rather less tangible than many products of industry. The challenge here is to try and look at educational systems through new eyes, to understand the value streams that underlie them. Womack (2006) defined lean education as three processes; designing, making and using. Breaking this down further, ‘designing’ means creating the knowledge to be delivered, ‘making’ means providing learning experiences for students, and ‘using’ means students being able to experience continuous learning. The question for educators is whether it is possible to make all these processes lean. Bringing such concepts right down to the classroom level, it is possible to assist students to appreciate lean thinking through practical learning activities. For example, Swanson (2008) describes a ‘lean lunch’ exercise, which helps students to understand the ‘Point-of-Use Staging’ technique, designed to reduce waste by shortening the supply chain. With a broader focus, Ncube (2010) outlines how the ‘Lean Lemonade Tycoon’ game can be used with students to help them to understand lean principles. From such small activities, lean thinking can be developed in students. According to Barney and Kirby (2004), educators can learn from lean production the importance of empowering teachers by training them to problem-solve and then expecting them to be self-reflective and to continuously improve their practice (many of these factors are also apparent in agile).



## 6 Reinterpreting Lean Thinking for Teaching and Learning

In section three, we identified specific strategies from agile methods that could be implemented in the classroom. From an agile perspective, we looked at values, processes and techniques. In this section, we analyse lean thinking using related categories; value, the value stream and perfection, lean processes and lean techniques.

### 6.1 Value, the Value Stream, and Perfection

Three of the original five lean principles outlined by the Lean Enterprise Institute are value, the value stream and perfection (Womack & Jones, 2003)—the others being pull and flow (discussed later). Value is both the end product and the chain of processes that deliver it. The value stream is each step in the value process, designed to be as efficient as possible in meeting customer expectations, while perfection is pursued through continuous improvement.

One of the key challenges of lean is the difficulty of knowing how to add value. Womack and Jones (2003) question why is it hard to start at the right place, to correctly define value. This is partly because most producers want to make what they are already making and partly because many customers only know how to ask for some variant of what they are already getting. Educators, too, tend to work with traditional views of teaching, and learners are also likely to expect the established, familiar model of learning. It is difficult for both types of stakeholder to see how they might redefine the value of education. An interesting point, made by Dahlgard and Østergaard (2000), is that one difference between mass production and education is that if there is a defect in mass production, then customers are likely to notice that defect (e.g. a fault in a product) quite quickly, whereas in education they may never notice the defect at all.

A lean approach to value and the value stream in education would aim to precisely specify the value of each learning experience and identify how it fits into the wider value stream, so that every step in the educational supply chain delivers value to the learner. To follow this approach, questions have to be continually asked, such as: Does this part of the curriculum deliver value? Does this form of assessment deliver value? Does this step in the enrolment process deliver value? The constant focus should be on how the educator delivers value to the learner.

The lean principle of perfection requires a constant focus on improvement. Improvements may be operational, administrative or strategic, but they must be clearly seen and demonstrated by satisfied customers (Womack & Jones, 2003). There are two types of improvement; *kaikaku* (radical improvement) and *kaizen* (continuous, incremental improvement). Bicheno (2001) discusses the differences between incremental and 'breakthrough' improvement, and suggest that that these need not always be driven by enforced processes but may also be passive, through

ongoing incentives such as quality circles. The main principle is to pursue perfection through continuous improvement of educational processes, methods and materials.

## 6.2 *Lean Processes*

One of the concepts of process that comes from mass production is the batch and queue approach (Poppendieck, 2011), and we tend to adopt this mass production view in many areas of education. Every semester/term we might deliver a set of classes over a fixed period of weeks, but this may have no relationship to how long it might take someone to actually learn something. Every year we produce a batch of graduating students, but often we do not know whether they are in fact ready to take advantage of the opportunities around them.

The two remaining principles of lean thinking are ‘flow’ and ‘pull’ (Womack & Jones, 2003). Flow replaces batch and queue processes that transform raw material into an end product. The goal is to provide continuous flow with minimal waste. In education, we aim to make learning flow without interruptions by right-sizing what is offered to the learner. Rather than broadcasting batches of content in mass production lecture halls or traditional classrooms, the lean educator would be engaged in the whole learning value stream, working closely with their colleagues across the whole process, moving away from batch blocks of material to smaller, more flexible learning components. A simple example of flow from a student perspective would be the ability to continue directly to the next stage of a course when they are ready, rather than waiting on institutional calendars (Isaksson, Kuttainen, & Garvare, 2013). In a similar vein, Alp (2001) suggests students being able to study at their own pace.

The principle of ‘pull’ means that the customer pulls the content they require, rather than having it pushed at them. The pull concept states that nothing should be built until a customer ‘pulls’ the product or service down the value stream (Womack & Jones, 2003). In education, content is often pushed towards the learner over timescales dictated by institutions. In a lean approach, the learner would pull from the educator the value they need, when they need it. One example where pull might be applied to education is by flexibly integrating demand from employers into vocational education programmes, giving students better prospects for employment on finishing their courses (Isaksson et al., 2013). Another idea is that students could create their own cross-disciplinary assignments to suit their needs (Alp, 2001). In a further example, Allen (2012) describes how pull was successfully used in an introductory Psychology class, allowing learners to take ownership of their own learning by choosing what they wanted to learn through discussion and priority setting.

At the highest level of lean processes there is the concept of the ‘lean enterprise’, which spans the whole value chain, and may involve many organisations (Womack & Jones, 2003). For a given school, the lean enterprise may involve the other schools that feed students into it, and the various schools, higher education institutions or workplaces that these students might move onto afterwards. All of these institutions are potentially part of the same lean enterprise. The difficult challenge is to convince

all of the stakeholders in that value chain to cooperate in becoming a single lean enterprise, by setting aside their own agendas. This would be a major task but is an important part of lean transformation. Godbey and Richter (1999) define the ‘agile-virtual organization’ as being cooperative, customising, fast and flexible, valuing human capital and relationships. Their description resonates strongly with the lean enterprise, with its vision of multiple institutions working together as a single enterprise. In education, they emphasise relationships over technology, while recognising the power of technology for collaboration and reach. As Comm and Mathaisel (2005) suggest, “Imagine a collaborative, higher education environment where duplicate functions do not exist but have shared resources with other institutions” (p. 236).

### 6.3 *Lean Techniques*

Pull and flow can be managed in a lean classroom by using Kanban (visual card) boards, which are one of the most widely used lean techniques, perhaps because of all the Japanese techniques it is the most exportable, relying little on its cultural context (Briggs, 1988). Kanban boards are similar to the agile storyboards mentioned earlier in this chapter, but the way that the user story cards are managed through the process is different because of the focus on pull and flow, concepts that are not specifically applied with agile storyboards. Within software development the Kanban board is a tool to support workflow management as it is used to visualise workflow, track work-in-progress (WIP) and embodies the pull system approach to manufacturing (Heikkilä, Paasivaara, & Lassenius, 2016; Goldman, 2009). The Kanban board is used to indicate progress in a transparent way and reinforces motivation and commitment to tasks. The Kanban board in education works in a very similar manner, but with more focus on learning flows or around tasks for assessment or learning activities. For example, it can be used in an individual or group activity to visually track student progress (Agile Classrooms, n.d.; ALC, n.d.). The Kanban board can be used to capture learning stories and then track progress in a visual manner and limit the WIP of an individual or student team. Examples of its use in education include an Education Kanban system used with trainee doctors (Goldman, 2009). The Kanban was used to record learning goals, identified in a collaborative manner, and record progress clearly and efficiently. The trainees were able to pull goals and work on them in a way that enabled them to take ownership of their own learning. In another example, at the Agile Learning Center, students use stand-up meetings alongside the use of Kanban boards to track their progress (ALC, n.d.).

Another key technique used in lean is the identification and methodical removal of ‘muda’ (waste). There are two types of muda. Type one muda is the waste caused by fixed components in the way that the system currently operates, for example an unwieldy student enrolment system, or a learning management system with inadequate functionality. In contrast, type two muda is waste that can be eliminated immediately. Of course, educators will encounter a high degree of type one muda. They tend to work within very bureaucratic organisations, sometimes very large ones,

with systems that are very difficult to change. Nevertheless, type one muda can be addressed by applying lean principles. A good example of how type one muda can be removed in education is outlined in Doman's (2011) study of a group of students who were introduced to lean principles and then tasked with re-engineering their institution's grade change process. They took a manual process that had evolved in a haphazard fashion over 50 years or so, and performed a lean analysis on how it could be improved. Their new online system design was adopted by their institution and proved to be both more efficient than the old system and to result in better outcomes for all stakeholders.

While addressing type one muda requires considerable effort and time, educators are also likely to encounter a large amount of type two muda that can be identified and removed from the value chain much more quickly and easily. The Toyota Production System includes seven categories of waste (Hines & Taylor, 2000). Since it is probably more appropriate to look at evaluating education as a service than a product, it may be helpful to examine these definitions as forms of service waste (adapted here from Bicheno & Holweg, 2000):

1. Overproduction—Are too many courses being offered, or is there too much information in classes, and is the right education being produced at the right time?
2. Defects—Are learning materials of good enough quality? Are there frequent errors in learning and administrative materials? Are classes well delivered (face to face or online)
3. Unnecessary inventory—Is there too much material in a learning management system or course? Does it take a long time to provide learning materials or give feedback? Are certain services half finished?
4. Inappropriate processing—Do people have to put in excessive effort in order to deliver a service? Do reports, exam results, etc. use over complex and slow systems?
5. Excessive Transportation—Do educators and/or students have to physically move themselves, or materials around? Does information move around unnecessarily?
6. Waiting—Do stakeholders have to wait a long time for information? Are processes unnecessarily delayed, waiting for something else to happen? Is information flowing correctly and efficiently?
7. Unnecessary motion—Are activities, paperwork and other efforts unnecessarily juggled?

Since the creation of the initial list of seven wastes, there have been a number of suggestions for adding further types of waste to the list as the '8th waste'. One of those adopted by a number of organisations is the waste of human talent—are skills being under-utilised? Are staff performing tasks that are not adding any value to the organisation? (Oppenheim, Murman & Secor, 2011). Using these ideas around the various forms of service waste, we may identify both type one and two muda. For example, Doman's (2011) study found type one muda with inappropriate processing and waiting, but any educational set of processes is also likely to reveal many examples of type two muda. There are probably many courses being offered where there

is too much material, for example, and where educators can easily begin to address this type of muda.

Another important lean technique is selecting and designing where students learn. Womack (2006) discusses the ‘gemba’ (the ‘real place’, sometimes written as ‘genba’), where problems are visible, and from where the best improvement ideas will come. In lean education, he suggests teaching process thinking and problem-solving by doing. With a focus on learning in real-world environments, he proposes developing a range of gemba for applied learning. This could be done by building relationships with industry. However, in many cases it may be hard to find suitable other organisational venues, so the learning institution can equally be used as a gemba. Perhaps by providing environments such as Makerspaces, schools and higher education providers can become more effective gemba. Böhmer, Beckmann, and Lindemann (2015) suggest that Makerspaces can be part of an open innovation ecosystem that embraces other agile and lean principles. In whichever physical context, it is important to grade students on the degree to which problems are actually solved using a rigorous method. This ties in closely with the concept of real-world learning and problem-solving as being a core twenty-first century skill (ITL Research, 2012). One example of using the learning context as a gemba and requiring students to solve problems within it is Marley’s (2014) description of students making lean-related videos as part of their assessment.

A related concept is the ‘gemba walk’, which means visiting the real place to identify problems and muda. As lean educators, it is therefore important to enable students to work in real places related to their learning, with a specific task to solve problems and/or identify waste. A clear example of this, referred to earlier in this chapter, was the students who used their own institution as the gemba and removed waste from the grade change system (Doman, 2011).

## 7 Making Learning Lean

Bringing these various lean ideas together, we might make some proposals about what lean learning might look like. There might be a curriculum delivered by multiple organisations, tailored to suit the learner, across a Lean Enterprise. There would be no restrictions on hours per week of learning, or the total time span of a learning process, or on how different learning components might be combined. Batch and queue delivery of learning would be replaced by flow, with walk-in, lifelong enrolment. There would be all possible combinations of blended learning modes, so providers of learning would have a pull approach rather than push, and there would be instant assessment feedback, for example, perhaps just one way in which we might think about removing inventory from our system.

## 8 Agile Education, Lean Learning

In this chapter, we have discussed various approaches to applying agile and lean methods to teaching and learning. In this closing section, we put the various concepts together to analyse how agile and lean methods can effectively transition from the contexts of software development and manufacturing into a coherent model for education. Figure 1 shows the major applications of agility and leanness in education that we have examined, structured around the concepts of values, processes and techniques. These concepts are expressed as a pyramid where techniques build on processes, and processes on values. At each of these levels we have examined how agile and lean practices may usefully transition to the context of teaching and learning.

First, we examine agile and lean values, which we summarise as agency, outcomes and improvement. One of the key concepts from the values and supporting principles of the Manifesto for Agile Software Development is that agile approaches give agency to both developers and customers. Giving students a similar sense of empowerment and agency over their own learning is an important challenge for twenty-first-century approaches to teaching and learning (Lindgren & McDaniel, 2012), while developing the broad skill sets encouraged in multi-skilled agile teams also reinforces learner agency. In agile software development, the key measure of progress is working software, and creating this requires well-developed technical skills. In education, the key measure of progress is meaningful learning, and in the twenty-first century this is primarily skills-based, not just technical skills but broader capabilities such as adaptability, collaboration, knowledge construction, real-world problem-solving

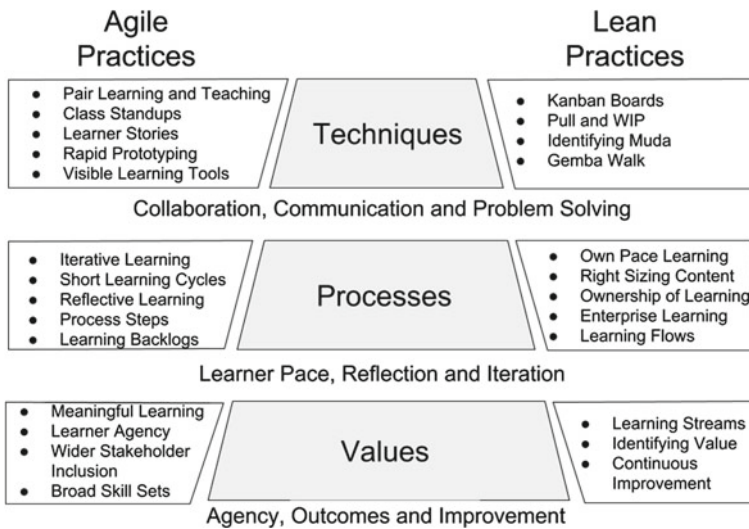


Fig. 1 Agile and lean values, processes and techniques, applied to learning

and innovation (ITL Research, 2012; World Economic Forum, 2016). Further, there is a broadening out of stakeholders in education to include learner's families and communities, so empowerment and agency can spread to these other stakeholders. From a lean perspective, it is important to identify the value and the value stream in the educational process, to identify the outcomes from each step in learning as well as from the overall experience. Underpinning this people-centric approach to skills development is the lean value of perfection; applying both kaizen and kaikaku to continually improve.

When we look at processes, we emphasise learner pace, reflection and iteration. Agile processes include regular evidence of progress and the ability to iterate over and reflect on solutions until they are as good as they can be. Short learning cycles, like sprints in agile software development, support a sustainable pace of learning, with regular feedback and reflection on actual learning outcomes. The steps in an agile process allow learners to address a specific learning backlog in manageable and self-directed time boxes. From lean, the concepts of pull and flow suggest that these processes need to be student-driven and adaptive to student needs by enabling learners to right-size content and learn at their own pace. Learners need to have ownership over their own learning so that they have control over how their learning value stream is constructed. Ideally, learning needs to be able to flow across an integrated learning path that spans multiple institutions in a lean educational enterprise. This might, for example, mean individual students in school having different personalised timetables, including learning sessions where they connect and collaborate with people beyond their physical environment to generate and critique new ideas (Starkey, 2012).

In examining techniques, it is difficult to focus on a particular subset and regard this as representative of agile and lean techniques as a whole, because there are so many. However, we consider that the most important techniques are those that support collaboration, communication and problem-solving. One of the agile techniques we have discussed in this chapter is pairing, which provides inbuilt peer support and collaborative learning. Similarly, class stand-ups provide collaborative peer support on a regular basis. Learner stories capture not just the 'what' of classroom activities but also the 'who' and the 'why', emphasising the need for relevance in learning activities. Students creating their own stories increase agency and self-regulation. In agile software development, prototyping means creating software prototypes. In the classroom, it can mean a broader set of activities including, for example, design thinking, with students being able to create prototypes not just with software but with 3D printing, craft materials, electronics, etc. However, the outcomes are the same; a deeper understanding of the customer and the product, not just from a business perspective but for social innovations such as creating alternative learning environments (Brown & Wyatt, 2010). In agile teams, information radiators such as burndown charts communicate visible records of progress. In the classroom, similar tools can be used to bring the same level of visibility to learning progress for both individuals and teams, echoing Hattie and Yates' (2013) emphasis on the importance of seeing outcomes from the learning process. The Kanban board is probably the most commonly used lean technique integrated into agile processes, adding the concepts of pull and work-in-progress to the management of story cards. Again, it is a practice that provides

students with agency over the content and pace of their own learning. It is probably also the most commonly used lean technique in the classroom, with many teachers using Kanban boards with their students (e.g. Beidleman, 2012). Another important lean technique that can work well in education is identifying wasteful practices (various types of muda) to try to remove these from the learning value stream and ensure that all activities that take place in the classroom and in the administration of learning are worthwhile. The final lean technique included here is contextualising learning in real-world problem spaces (gemba). As one of the six twenty-first-century skills outlined by ITL Research (2012), real-world problem-solving and innovation is a key activity in contemporary learning. The concept of the gemba walk, where problems are identified and resolved in their real-world contexts, supports this skill development and also relates closely to the learning theory of situated cognition, whereby knowledge “indexes the situations in which it arises and is used, without which it cannot be fully understood” (Brown, Collins, & Duguid, 1989, p. 16).

## 9 Conclusion

In this chapter we have provided a brief summary of the various themes and ideas gathered from the literature on agile and lean approaches to education. In Fig. 1 and the associated commentary we have brought together what we see as the most important ideas from the relevant literature, in order to provide a simple overview of how agile and lean approaches to teaching and learning can be integrated into new ways of thinking about what goes on in the classroom and in the wider world of learning.

In conclusion, we believe that there are significant opportunities for educators to adopt aspects of agile and lean practices. Although some interesting work has already been undertaken in this area, our investigation of the literature suggests that there is much more that could be done to bring the benefits of agility and lean thinking into the classroom. In this chapter, we have attempted to provide a clearer understanding of the ways that agile and lean approaches can be applied to teaching and learning at three specific levels; values, techniques and processes, and we have summarised the core skills and practices that we believe are the essential components in reinterpreting agile and lean concepts for teaching and learning. These are the values of agency, outcomes and (continuous) improvement, processes for reflection and iteration at a pace managed by the learner, and techniques for collaboration, communication and (real world) problem-solving.



## References

- ALC. (n.d.). *Agile learning center tools & practices*. Retrieved from <http://nycagile.org/about/tools/>.
- Alp, N. (2001, November). The lean transformation model for the education system. In *Proceedings of the 29th International Conference on Computers and Industrial Engineering* (pp. 82–87).
- Agile Alliance. (2001). *Manifesto for agile software development*. Retrieved from <http://agilemanifesto.org/>.
- Agile Classrooms. (n.d.). *21st century learning resources*. Retrieved from <https://www.agileclassrooms.com/resources>.
- Alves, A. C., Flumerfelt, S., & Kahlen, F. J. (Eds.). (2016). *Lean education: An overview of current issues*. Springer.
- Anderson, D. (2010). *Kanban: Successful evolutionary change for your technology business*. Sequim, WA: Blue Hole Press.
- Allen, M. (2012). *Leaving ADDIE for SAM: An agile model for developing the best learning experiences*. ASTD Press.
- Balzer, W. K. (2010). *Lean higher education: Increasing the value and performance of university processes*. CRC Press.
- Barney, H., & Kirby, S. N. (2004). Toyota production system/lean manufacturing. In B. Stecher & S. N. Kirby (Eds.), *Organizational improvement and accountability lessons for education from other sectors* (pp. 35–50). Santa Monica, CA: Rand Corporation.
- Beck, K., & Andres. C. (2005). *Extreme programming explained: Embrace change* (2nd ed.). Addison-Wesley.
- Bicheno, J. (2001). Kaizen and kaikaku. In *Manufacturing operations and supply chain management: The LEAN approach* (pp. 175–184).
- Bicheno, J., & Holweg, M. (2000). *The lean toolbox* (Vol. 4). Buckingham: PICSIE Books.
- Beidleman, P. (2012). Using Kanban in the Classroom. *Planview LeanKit*. Retrieved from <https://leankit.com/blog/2012/01/guest-post-using-kanban-in-the-classroom/>.
- Böhmer, A. I., Beckmann, A., & Lindemann, U. (2015, December). Open innovation ecosystem-makerspaces within an agile innovation process. In *ISPIM Innovation Symposium. The International Society for Professional Innovation Management (ISPIM)* (p. 1).
- Briggs, P. (1988). The Japanese at work: Illusions of the ideal. *Industrial Relations Journal*, 19(1), 24–30.
- Briggs, S. (2014). Agile based learning: What is it and how can it change education? *InformED*. Retrieved from <http://www.opencolleges.edu.au/informed/features/agile-based-learning-what-is-it-and-how-can-it-change-education/>.
- Brown, J. S., Collins, A., & Duguid, P. (1989). Situated cognition and the culture of learning. *Educational researcher*, 18(1), 32–42.
- Brown, T., & Wyatt, J. (2010). Design thinking for social innovation. *Development Outreach*, 12(1), 29–43.
- Bruegge, B., Reiss, M., & Schiller, J. (2009, April). Agile principles in academic education: A case study. In *Sixth International Conference on Information Technology: New Generations, 2009. ITNG'09* (pp. 1684–1686). IEEE.
- Cohen, D., Lindvall, M., & Costa, P. (2004). An introduction to agile methods. *Advances in Computers*, 62, 1–66.
- Comm, C. L., & Mathaisel, D. F. (2005). An exploratory study of best lean sustainability practices in higher education. *Quality Assurance in Education*, 13(3), 227–240.
- Conboy, K. (2009). Agility from first principles: Reconstructing the concept of agility in information systems development. *Information Systems Research*, 20(3), 329–354. <https://doi.org/10.1287/isre.1090.0236>.
- Dahlgard, J. J., & Østergaard, P. (2000). TQM and lean thinking in higher education. *The Best on Quality: Targets, Improvements, Systems*, 11, 203–226.

- Delhij, A., et al. (2016). *Agile in education compass*. Retrieved from <http://www.agileineducation.org/>.
- Dewi, D. A., & Muniandy, M. (2014, September). The agility of agile methodology for teaching and learning activities. In *2014 8th Malaysian Software Engineering Conference (MySEC)* (pp. 255–259). IEEE.
- Dingsøy, T., Nerur, S., Balijepally, V., & Moe, N. B. (2012). A decade of agile methodologies: Towards explaining agile software development. *Journal of Systems and Software*, *85*(6), 1213–1221. <https://doi.org/10.1016/j.jss.2012.02.033>.
- Doman, M. S. (2011). A new lean paradigm in higher education: A case study. *Quality Assurance in Education*, *19*(3), 248–262.
- Francis, D. E. (2014). Lean and the learning organization in higher education. *Canadian Journal of Educational Administration and Policy*, *157*, 1–23.
- Godbey, G. C., & Richter, G. J. (1999). Technology, consortia, and the relationship revolution in education. *New Directions for Higher Education*, *1999*(106), 85–91.
- Goldman, S. (2009). The educational Kanban: Promoting effective self-directed adult learning in medical education. *Academic Medicine*, *84*(7), 927–934.
- Hattie, J., & Yates, G. C. (2013). *Visible learning and the science of how we learn*. Routledge.
- Hazzan, O., & Dubinsky, Y. (2014). *Agile anywhere: Essays on agile projects and beyond*. Springer International Publishing.
- Hew, K. F., & Brush, T. (2007). Integrating technology into K-12 teaching and learning: Current knowledge gaps and recommendations for future research. *Educational Technology Research and Development*, *55*(3), 223–252.
- Heikkilä, V. T., Paasivaara, M., & Lassenius, C. (2016, May). Teaching university students Kanban with a collaborative board game. In *Proceedings of the 38th International Conference on Software Engineering Companion* (pp. 471–480). ACM.
- Hines, P., & Taylor, D. (2000). *Going lean* (pp. 3–43). Cardiff, UK: Lean Enterprise Research Centre Cardiff Business School.
- Isaksson, R., Kuttainen, C., & Garvare, R. (2013). Lean higher education and lean research. In *16th Toulon-Verona Conference; Faculty of Administration, University of Ljubljana, Slovenia; 29–30 August 2013*.
- ITL Research. (2012). *21CLD learning activity rubrics*. Retrieved from <https://education.microsoft.com/GetTrained/ITL-Research>.
- Kamat, V. (2012, July). Agile manifesto in higher education. In *IEEE Fourth International Conference on Technology for Education (T4E)* (pp. 231–232). IEEE.
- Kessler, R., & Dykman, N. (2007). Integrating traditional and agile processes in the classroom. *ACM SIGCSE Bulletin*, *39*(1), 312–316.
- Krehbiel, T. C., et al. (2017). Agile manifesto for teaching and learning. *The Journal of Effective Teaching*, *17*(2), 90–111.
- Kropp, M., Meier, A., Mateescu, M., & Zahn, C. (2014). Teaching and learning agile collaboration. In *IEEE 27th Conference on Software Engineering Education and Training (CSEE&T)*.
- Ladas, C. (2008). *Scrumban: Essays on Kanban systems for lean software development*. Seattle, WA: Modus Cooperandi.
- Layman, L., Cornwell, T., & Williams, L. (2006). Personality types, learning styles, and an agile approach to software engineering education. *ACM SIGCSE Bulletin*, *38*(1), 428–432.
- Lindgren, R., & McDaniel, R. (2012). Transforming online learning through narrative and student agency. *Educational Technology & Society*, *15*(4), 344–355.
- Marley, K. A. (2014). Eye on the gemba: Using student-created videos and the revised bloom's taxonomy to teach lean management. *Journal of Education for Business*, *89*(6), 310–316.
- Meerbaum-Salant, O., & Hazzan, O. (2010). An agile constructionist mentoring methodology for software projects in the high school. *ACM Transactions on Computing Education*, *9*(4), n4.
- Mahnic, V. (2012). A capstone course on agile software development using Scrum. *IEEE Transactions on Education*, *55*(1), 99–106.

- Manamendra, M. A. S. C., Manathunga, K. N., Perera, K. H. D., & Kodagoda, N. (2013, April). Improvements for agile manifesto and make agile applicable for undergraduate research projects. In *2013 8th International Conference on Computer Science & Education (ICCSE)* (pp. 539–544). IEEE.
- Melnik, G., & Maurer, F. (2003, August). Introducing agile methods in learning environments: Lessons learned. In *Conference on Extreme Programming and Agile Methods* (pp. 172–184). Berlin, Heidelberg: Springer.
- Naylor, J. B., Naim, M. M., & Berry, D. (1999). Leagility: Integrating the lean and agile manufacturing paradigms in the total supply chain. *International Journal of Production Economics*, *62*(1–2), 107–118.
- Ncube, L. B. (2010). A simulation of lean manufacturing: The lean lemonade tycoon 2. *Simulation & Gaming*, *41*(4), 568–586.
- Obrist, M., Moser, C., Fuchsberger, V., Tscheligi, M., Markopoulos, P., & Hofstätter, J. (2011, June). Opportunities and challenges when designing and developing with kids@ school. In *Proceedings 10th International Conference on Interaction Design and Children* (pp. 264–267). ACM.
- Oppenheim, B. W., Murman, E. M., & Secor, D. A. (2011). Lean enablers for systems engineering. *Systems Engineering*, *14*(1), 29–55.
- Parsons, D., Ryu, H., & Lal, R. (2007). The impact of methods and techniques on outcomes from agile software development projects. In T. McMaster, D. Wastell, E. Ferneley, & J. DeGross (Eds.), *Organisational dynamics of technology-based innovation: Diversifying the research agenda, Proceedings of IFIP 8.6 Conference, Manchester, UK, 14–16 June 2007* (pp. 235–249). Springer.
- Peha, S. (2011, June). Agile schools: How technology saves education (Just not the way we thought it would). *InfoQ*. Retrieved from <https://www.infoq.com/articles/agile-schools-education>.
- Petersen, K. (2011). Is lean agile and agile lean?: A comparison between two software development paradigms. In A. Dogru & V. Biçer (Eds.), *Modern software engineering concepts and practices: Advanced approaches* (pp. 19–46). IGI Global.
- Poppendieck, M. (2011). Principles of lean thinking. *IT Management Select*, *18*, 1–7.
- Poppendieck, M., & Poppendieck, T. (2003). *Lean software development: An agile toolkit*. Boston MASS: Addison-Wesley Professional.
- Rico, D. F., & Sayani, H. H. (2009, August). Use of agile methods in software engineering education. In *Proceedings Agile Conference, 2009. AGILE'09*. (pp. 174–179). IEEE.
- Scrum@School. (n.d). *Advantages*. Retrieved from <http://www.scrumatschool.co.uk/>.
- Starkey, L. (2012). *Teaching and learning in the digital age*. Abingdon, Oxon: Routledge.
- Stewart, J. C., DeCusatis, C. S., Kidder, K., Massi, J. R., & Anne, K. M. (2009). Evaluating agile principles in active and cooperative learning. In *Proceedings of Student-Faculty Research Day, CSIS, Pace University*, B3.
- Swanson, L. (2008). The lean lunch. *Decision Sciences Journal of Innovative Education*, *6*(1), 153–157.
- Tesar, M., & Sieber, S. (2010). Managing blended learning scenarios by using agile e-learning development. In *Proceedings of the IADIS International Conference E-Learning* (Vol. 2, pp. 125–129).
- Vanhoeacker, M. (2015). *Skyfaring: A journey with a pilot*. London: Chatto & Windus.
- Womack, J. P., Jones, D. T., & Roos, D. (1990). *The machine that changed the world*. New York, NY: Free Press.
- Womack, J. P., & Jones, D. T. (2003). *Lean thinking: Banish waste and create wealth in your corporation* (2nd ed.). New York, NY: Free Press.
- Womack, J. P. (2006). *Lean thinking for education*. Paper Presented at the 2006 Lean Educator Conference, Worcester, MA.
- World Economic Forum. (2016). *New vision for education: Fostering social and emotional learning through technology*. Retrieved from [http://www3.weforum.org/docs/WEF\\_New\\_Vision\\_for\\_Education.pdf](http://www3.weforum.org/docs/WEF_New_Vision_for_Education.pdf).
- Zion, S. D. (2009). Systems, stakeholders, and students: Including students in school reform. *Improving Schools*, *12*(2), 131–143.

# Agile Methodologies in Education: A Review



Pasquale Salza, Paolo Musmarra and Filomena Ferrucci

**Abstract** One of the main challenges faced by teachers in education, both at K-12 and academy levels, is related to the need to attract and retain the attention and the commitment by students, and ensure they achieve the required learning outcomes. Thus, new and exciting methodologies were developed to support teachers. Many of them have been inspired by approaches devised for Agile software development. This chapter aims to review the main Agile methodologies that have inspired educational approaches and to provide a description of the features retained in the educational context. Several experiences reported in the literature are also described.

**Keywords** Agile · Education · Review · eXtreme Programming · Scrum

## 1 Introduction

Agile is one of the most used process frameworks for software development. It is based on some essential values and principles, i.e., the Agile Manifesto, with the aim at lightening the traditional and linear waterfall approach to face the real world in which requirements and solutions evolve continuously (Beck et al., 2001). Agile favors an iterative and team-based approach, attempting to reduce the waste of resources, development time and effort. Several methodologies have been defined within the Agile culture, extending and implementing its values and principles, such as eXtreme Programming, and Scrum.

---

P. Salza (✉)  
USI Università della Svizzera italiana, Lugano, Switzerland  
e-mail: [pasquale.salza@usi.ch](mailto:pasquale.salza@usi.ch)

P. Musmarra · F. Ferrucci  
University of Salerno, Fisciano, Italy  
e-mail: [pmusmarra@unisa.it](mailto:pmusmarra@unisa.it)

F. Ferrucci  
e-mail: [fferrucci@unisa.it](mailto:fferrucci@unisa.it)

One of the most relevant implications to managers working using Agile is that it places more emphasis on people factors, focusing on the talents and skills of individuals. If people on a project are good enough, they can use almost any process and accomplish their assignment. Agile makes people work together with excellent communication and interaction, using their individual talents in teams to reach common goals efficiently (Cockburn & Highsmith, 2001).

Agile places less focus on fixed and well detailed a priori plans and is based on the empirical process control theory suggesting that significant knowledge comes from what we learn through experience. An iterative development approach is exploited aiming to deliver product increments that provide value to the customer.

Since Agile methodologies have proved very useful in managing software development teams and projects, the intuitions of many researchers was to adapt them to the educational context (Dewi & Muniandy, 2014). Agile methodologies were first introduced as part of software engineering courses, where the teacher manages student teams making them practice in real software projects (Alfonso & Botia, 2005). Then, Agile methodologies proved to be also effective in teaching other subjects, e.g., Mathematics (Duvall, Hutchings, & Kleckner, 2017).

This chapter aims to give an overview of those Agile methodologies for which some research work, describing the adaptation to the classroom environment, is present in the literature. The aim is to highlight Agile values, principles, and practices applied to improve students' learning. Our review does not aim to be complete and systematic but a starting point for researchers and educators.

The rest of the chapter is organized as follows. Section 2 presents the literature search phase with some statistics on the selected pool of publications. Section 3 introduces Agile, describing the motivations, values, and principles that led it to be one of the most popular software development processes. eXtreme Programming is described in Sect. 4 and Scrum in Sect. 5. Each of these sections are organized as follows. The history and key points of the methodology are first provided within the original software development context. Then, the focus is placed on the education environment and the most employed adaptations in the literature are given. Each section ends with an overview of some of the most relevant, or recent, experiences in the literature. Section 6 concludes the chapter with some final remarks.

## 2 Search Strategy

In this section, we present an overview of our research method, the goals of the review and some general statistics about the findings. In terms of timeline, the searches were conducted during late 2017.

Even though this chapter cannot be technically defined as a Systematic Literature Mapping or Systematic Literature Review, we partially developed our search strategy based on well-known guidelines (Kitchenham & Charters, 2007). These were used to organize the research phase and structure the contents of the chapter.

## 2.1 Search Goals

The purpose of this study is to introduce agile concepts and connect them to the classroom environment, giving a first overview to the readers who are interested in starting to investigate the field.

We identified the following main search goals:

- organize the Agile methodologies with respect to their application in the classroom environment;
- identify the main trends of Agile methodologies for teaching and learning;
- discover what are the main education levels, e.g., K-12, academy, investigated in the literature;
- find out if these methodologies are employed for online, other than on-site education;
- understand if the Agile methodologies for education have gone beyond the software engineering education field to be used to teach and learn other subjects.

## 2.2 Source Engines and Search Keywords

We employed the main sources of relevance for scientific publications in the field of Computer Science:

- IEEExplore;
- ACM Digital Library;
- ScienceDirect;
- Scopus;
- Google Scholar.

We did not use only the Google Scholar aggregator since several sources have mentioned that “it should not be used alone” as it may miss some sources (Haddaway, Collins, Coughlin, & Kirk, 2015). Indeed, this was something that we easily verified during our search.

We used a query composed by terms related to the Agile context and methodologies, mixed with terms related to the education field. Specifically, our query was expressed by

(agile OR scrum OR kanban OR extreme programming OR pair programming) AND (education OR teaching OR learning OR classroom OR K-12)

Even though it was not systematic, we tried to include all the relevant sources as much as possible using the snowballing technique (Wohlin, 2014). Snowballing, in this context, is based on the use of the reference list or citations to a paper to identify additional papers.

We collected only the papers written in English and published from 2003 to 2017 in international journals and the proceedings of international conferences.

### 2.3 Selected Papers

From our search, we found a total of about 200 papers. Nevertheless, we did not use all of them to compose this chapter. We only selected those we considered as the most relevant to represent the concepts and goals of our study. In this section, we give some statistics about them.

The majority of the papers, i.e., 80.5%, were published in the proceedings of international conferences. 18% was composed of publications in international journals. The rest consisted of one book chapter and one technical report.

In terms of date of publication, 36 papers (18%) were published from 2003 to 2009. Starting from 2010, the number increased significantly year by year, from 14 to 27 in 2017 for a total of 164 papers in that range of years (82%).

As for the methodologies, we identified some main categories

- Agile: 83.5% of the papers refer generically to “Agile” as a methodology for teaching and learning;
- Scrum: 41% of the papers refer specifically to Scrum;
- eXtreme Programming: 11% of the papers refer specifically to eXtreme Programming;
- Pair Programming: 19.5% of the papers refer specifically to Pair Programming. The number is higher than for eXtreme Programming since it includes many works considering “Pair Programming” as an independent methodology;
- Kanban: only 3% of the papers refer specifically to Kanban.

We also divided the papers by education level. Very few of them, i.e., 10%, are papers targeting K-12 students, from a minimum of 4- to 19-year-old (the ranges can change according to different countries). The rest is focused on academy students, where 87.5% is for undergraduates and 18.5% specifically for master students.

The majority of the papers refer to on-site education. However, we found 7 papers that address the adaptation of Agile methodologies within the e-learning context.

Finally, we distinguished between works that use Agile methodologies for teaching and learning specific subjects. As expected, the majority of them, i.e., 91.5%, are about Computer Science and Software Engineering, since it is easy to adapt methodologies originally devised for software development by reproducing a subset of concepts for the classroom environment. However, 8.5% refer also to different fields, e.g., Mathematics.

## 3 Agile

In 2001, a team of 17 leading software practitioners devised and published what they defined the “Manifesto for Agile Software Development”. It is a document that defines the values and principles of the Agile software development movement as a summary of how they found “better ways of developing software by doing it and helping others to do it” (Beck et al., 2001).

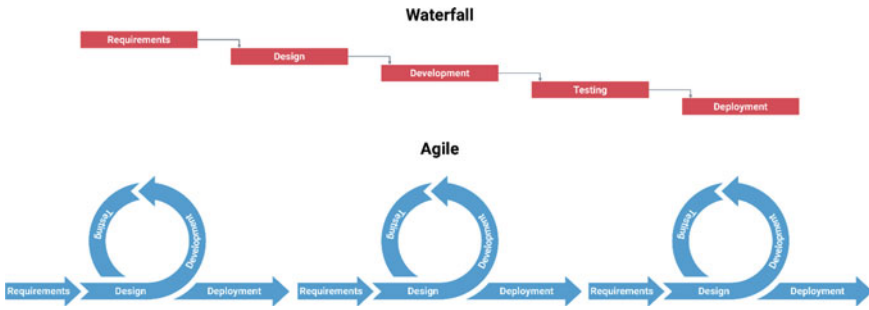


Fig. 1 The waterfall and Agile lifecycles

The key idea is to have an incremental and iterative approach instead of an in-depth planning at the beginning of a software project. Agile methodologies are open to changes in requirements and encourage constant feedback from end users/customers.

In an Agile lifecycle, shown in Fig. 1, there is not a strict sequence of events to follow as the classic waterfall model has. The phases are flexible and always evolving, and many of them can even happen in parallel:

1. *requirements analysis*: involves many meetings with the managers, stakeholders, and users to identify the business requirements. The team collects quantifiable, relevant, and detailed information, i.e., who will use the product, and how;
2. *planning*: once the idea becomes viable and feasible, the team splits it into smaller pieces of work, prioritizing and assigning them to different iterations;
3. *design*: the team looks for a solution for the requirements, together with a test strategy;
4. *development*: the features are implemented;
5. *testing*: the produced code is tested against the requirements to make sure the software is solving the customer needs;
6. *deployment*: at this point, the product is delivered to the customers to be used, but this is not the end of the project. It can be only a partial delivery, and new requirements could come.

Agile has given birth to several methodologies, all following its philosophy (Abrahamsson, Salo, Ronkainen, & Warsta, 2002; Dingsøy, Nerur, Balijepally, & Moe, 2012; Dybå & Dingsøy, 2008). Table 1 shows the main employed ones.

The table shows only a few of the available Agile methodologies. Indeed, industries do not necessarily follow their rules in a strict way and this produced many variants, or hybrids, e.g., Scrumban (Ladas, 2009). In the following sections, an overview of Agile values and principles adapted to the classroom environment is first given. Then, the focus is shifted to eXtreme Programming and Scrum since they represent the most relevant applications in the field of education.



**Table 1** Agile methodologies

Methodology	Description
Adaptive Software Development (ASD)	It focuses mainly on the problems in developing complex and large systems. The method strongly encourages incremental, iterative development, with constant prototyping. Its aim is to provide a framework with enough guidance to prevent projects from falling into chaos, but not too much, which could suppress emergence and creativity (Abrahamsson et al., 2002)
Crystal methods	A family of methods for co-located teams of different sizes and criticality: Clear, Yellow, Orange, Red, and Blue. Crystal Clear, the most used one, focuses on communication in small teams developing software that is not life-critical. Clear development has seven characteristics: frequent delivery, reflective improvement, osmotic communication, personal safety, focus, easy access to expert users, and requirements for technical environment (Cockburn, 2004; Dybå & Dingsøy, 2008)
Dynamic Software Development Method (DSDM)	It divides projects into three phases: pre-project, project lifecycle, and post project. Nine principles are present: user involvement, empowering the project team, frequent delivery, addressing current business needs, iterative and incremental development, allowing reversing changes, high-level scope being fixed before project starts, testing throughout the lifecycle, and efficient and effective communication (Dybå & Dingsøy, 2008; Stapleton, 2003)
eXtreme Programming (XP)	It focuses on best practices for development: the planning game, small releases, metaphor, simple design, testing, refactoring, Pair Programming, collective ownership, Continuous Integration, 40-hour workweek, on-site customers, and coding standards (Beck, 1999; Dybå & Dingsøy, 2008)
Feature-Driven Development (FDD)	It combines model-driven and Agile development with emphasis on the initial object model, a division of work in features, and iterative design for each feature. FDD claims to be suitable for the development of critical systems. An iteration of a feature consists of two phases: design and development (Dybå & Dingsøy, 2008; Palmer & Felsing, 2001)
Kanban	It is a scheduling method developed by Toyota for Lean production. It was designed as a system for scheduling to facilitate the production and inventory control. Using Kanban, work teams can achieve a Just-In-Time (JIT) manufacturing, reducing flow times within production system and response times from suppliers and to customers (Sugimori, Kusunoki, Cho, & Uchikawa, 1977)
Scrum	It focuses on project management in situations where it is difficult to plan, with mechanisms for “empirical process control”. The feedback loops constitute the core element. The software is developed by a self-organizing team in increments called “sprints”, starting with planning and ending with a final review. Then, the product owner decides which backlog items should be developed in the next sprint. Team members coordinate their work in a daily stand-up meeting. One team member, the Scrum master, is in charge of solving problems that might stop the team from work efficiently (Dybå & Dingsøy, 2008; Schwaber & Beedle, 2002; Schwaber & Sutherland, 2011)

### 3.1 Agile in Education

Many researchers had the intuition of tailoring Agile methodologies to the education environment (Dewi & Muniandy, 2014). Stewart et al. presented a first review of the literature aimed at showing how agile methods were applied to education (Stewart, DeCusatis, Kidder, Massi, & Anne, 2009). Moreover, they provided a mapping between values and principles of the Agile Manifesto to specific educational methods and activities.

Table 2 shows the values defined in the Agile Manifesto and the applied mapping, which consists of the translation of software development figures and roles to the education environment (Stewart et al., 2009).

As it is shown in the table, the first value favors student-centric environments as the most effective method of learning. In the Agile school, the old-fashioned lecture-driven environment is considered as surpassed. The students participate actively in the learning process through activities and group-based components aiming at reinforcing concepts and allowing for exploration.

As with software in Agile, the second value in education favors the production of working projects from the beginning, without waiting for the end of a project-based course. The students work in an iterative environment with a strong deliverable component, leading to higher immersion in the project, more learning, and final deliverables of better quality.

The third value allows having an environment focused on what the students are doing and which pedagogical methods can facilitate the learning. In traditional courses, the syllabus is outlined as a strict contract between the student and the instructor. Within the Agile school, the students and instructor can establish a more flexible and collaborative relationship, similar to the way that developers and customers collaborate following the Agile approach.

Finally, with the fourth value, agility is applied so that different learning approaches can be adopted and delivery methods changed if the current methods are not producing the expected results.

Table 3 shows the original principles of the Agile Manifesto and the mapping that Stewart et al. applied to the context of the educational environment. In line with

**Table 2** Mapping Agile values to the classroom environment (Stewart et al., 2009)

Value	Agile Manifesto	Agile Manifesto in Education
1	Individuals and interactions, over process and tools	Students over traditional processes and tools
2	Working software, over comprehensive documentation	Working projects over comprehensive documentation
3	Customer collaboration, over contract negotiation	Student and instructor collaboration over rigid course syllabi
4	Responding to change, over following a plan	Responding to feedback rather than following a plan

**Table 3** Mapping Agile principles to the classroom environment (Stewart et al., 2009)

Principle	Agile Manifesto	Agile Manifesto in Education
1	High priority to the customer satisfaction, early and continuously delivering valuable software	High priority to prepare the student to be self-organized, continuously delivering course components that reflect competence
2	Requirements can change at any time for the customer's competitive advantage	The instructor and students can adapt to changes at any time to facilitate learning and better develop marketable skills
3	Deliver working software frequently with a preference to the shorter timescale	Working deliverables from the students over short time periods allowing for frequent feedback
4	Business people and developers must work together daily	Iterative interaction between the instructor and students (or student groups)
5	Build projects around motivated individuals and support them in a proper environment	Give students the environment and support necessary to be successful
6	Prefer face-to-face conversation	Allow for direct face-to-face interaction with students or student groups
7	Working software is the primary measure of progress	Working deliverables (e.g., models, software, project deliverables, presentations) are the primary measure of student progress
8	The sponsors, developers, and users should be able to maintain a constant pace indefinitely	The cooperative learning environment is the basis for teaching the skills needed for life-long learning
9	Continuous attention to technical excellence and good design enhances agility	Continuous attention to technical excellence and good design enhances learning
10	Simplicity is essential	Understanding the problem and solving it simply and clearly is essential
11	The best architectures, requirements, and designs emerge from self-organizing teams	Student groups and teams should self-organize, but all should participate equally in the effort
12	At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly	At regular intervals, the students and instructor reflect and offer feedback on how to be more effective, then the stakeholders adjust accordingly to be more efficient

the values described above, the 12 principles highlight that the highest priority of Agile instructors is to satisfy the needs of the students, together with their families who become an active part in the learning process, through the early and continuous delivery of meaningful learning. Every change in this direction, even late in the learning cycle, is always welcome.

The students are motivated individuals, working in an environment based on the complete trust from the instructors about getting the job done. Indeed, the old-fashioned face-to-face conversation is assumed superior over technology, even though the latter is considered of undeniable importance and usefulness.

In this context, working deliverables constitute the most tangible measure of student progress. For this reason, meaningful and project-based learning is primarily encouraged with continuous attention to technical excellence and good design but promoting the simplicity of solutions. The students are self-organized in teams, all collaborating with the same effort and reflecting at regular intervals on how to become more effective, tuning and adjusting their behavior accordingly.

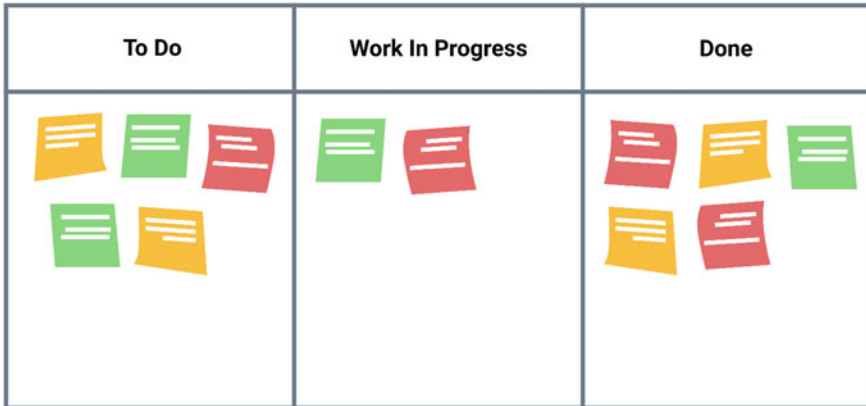
Agile has been effectively used in the academy to teach software engineering, mostly with a project-based learning approach where the final learning product is a software produced simulating the Agile practices in small groups (Alfonso & Botia, 2005; Bruegge, Reiss, & Schiller, 2009; Hanks, 2007; Kessler & Dykman, 2007; Lu & DeClue, 2011; Paez, 2017; Rico & Sayani, 2009). As an example, Alfonso and Botia described the use of an iterative and Agile process model in a software engineering course for undergraduate students (Alfonso & Botia, 2005). It served both as an educational technique for teachers and a subject of learning for students. The model allowed to the expected knowledge to be provided efficiently, acquired gradually, evaluated, and the learning process to be effectively driven.

The use of Agile in K-12 environments is also documented (Fronza, Ioini, & Corral, 2017; Kastl, Kiesmüller, & Romeike, 2016; Meerbaum-Salant & Hazzan, 2010; Romeike & Göttel, 2012). As a relevant example, Fronza et al. described the design and implementation of an educational framework using animations and Scratch to teach computational thinking skills based on Agile practices (Fronza et al., 2017). The framework covers 60 h, 4 h per week. The students start from brainstorming and produce storyboards and mindmaps. In each Agile iteration, they draw and program, checking the conformance with the requirements. During the retrospective phase, they analyze the whole project and plan future activities.

Some works report the effectiveness of applying Agile methodologies also in online courses (Ashraf, Shama, & Rana, 2012; Noguera, Guerrero-Roldán, & Masó, 2018; Vivian, Falkner, & Falkner, 2013). For instance, Noguera et al. proposed an approach for implementing the Agile method into online academic education context (Noguera et al., 2018). The results revealed that Agile strategies incorporated in project-based learning facilitated both the team regulation and project management. The teacher acted as a supervisor and facilitator, helping students improve their learning process iteratively through the development of the projects.

Agile was also employed to teach other subjects. Seman et al. reported a study about applying Agile to two project-based learning courses in electrical engineering (Seman, Hausmann, & Bezerra, 2018). The results showed the importance of the humanization feature in learning, automatically given by applying Agile, as a fundamental part of the education process in electrical engineering.

Finally, the literature reports many cases in which Agile was exploited for its useful tools, not necessarily applying a full methodology. An example is the Kanban board, especially in project-based learning (Ahmad, Liukkunen, & Markkula, 2014;



**Fig. 2** A typical Kanban board

Bacea, Ciupe, & Meza, 2017; Heikkilä, Paasivaara, & Lassenius, 2016). Widely used in industry, Kanban boards are a physical or electronic visualization tool for the management of work in teams, to improve their delivery of products and services in terms of predictability, quality and just-in-time performance. Figure 2 depicts a typical Kanban board, structured on multiple columns to visualize the workflow process.

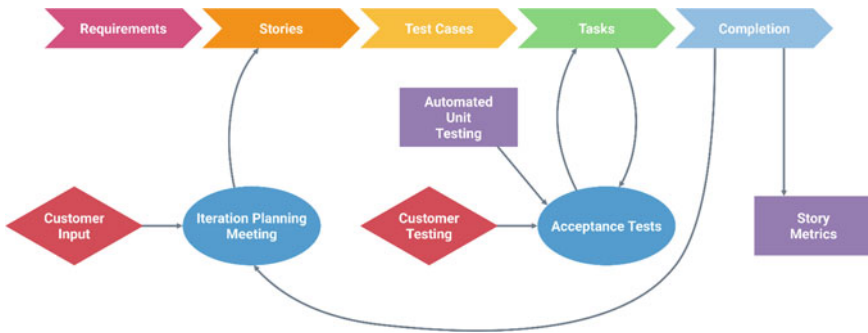
Every task in a project, represented by a card, will follow a path:

1. from the “to do” status, where the tasks are discussed within the team and then assigned to specific members;
2. passing to the “in progress” status when the tasks are being executed;
3. ending in “done” when completed.

## 4 eXtreme Programming

The origin of eXtreme Programming (XP) started in 1996 from Kent Beck, one of the 17 Agile practitioners that signed the Agile Manifesto. At that time, Beck was handling the Chrysler Comprehensive Compensation System (C3) project in the Chrysler Corporation aiming at replacing several payroll applications with a single system. In 1999, he collected all the experience at Chrysler in a book called “eXtreme Programming Explained: Embrace Change” (Beck, 1999).

XP is a development methodology that is intended to improve software quality and responsiveness to change in customer requirements. Figure 3 shows the XP lifecycle. XP allows frequent releases in short development cycles, improving the productivity of the team, and at the end of which new customer requirements can be adopted. XP employs user stories and associates acceptance tests to them that need to be successfully passed before the stories can be considered as done. Moreover,



**Fig. 3** The eXtreme Programming lifecycle

the programmer is expected to write tests for the individual tasks that contribute to a story. Indeed, XP puts tests before code, and each piece of code is expected to be associated with a test, or it should not be integrated.

XP is based on five values:

- *simplicity*: simple solutions are considered as cheaper and faster to be implemented than more complex solutions;
- *communication*: the documentation is always after the direct communication. All the team members should intensively communicate with each other through personal dialogue, aiming at avoiding misunderstandings;
- *feedback*: the customer is interviewed using very short feedback loops. The development progress is frequently shown, and mistakes can be thus avoided. The feedback also comes from the tests;
- *respect*: every member of the team is valuable, even if it is just enthusiasm;
- *courage*: using the XP values requires a great deal of courage. It is important to tell the truth about progress and estimates. Also, XP does not allow fear of refactoring old code or throwing it away if needed for changes.

A typical XP team includes six roles:

- *customer*: the person responsible for writing the user stories, setting priorities and specifying the functional tests;
- *programmer*: an ordinary developer who produces the code and performs the whole amount of expected project tasks;
- *coach*: watches and manages the teamwork, teaching its members how to implement the most effective practices;
- *tracker*: the person who monitors the progress of the software development and detects possible issues;
- *tester*: responsible for product testing;
- *doomsayer*: tracks the project risks and warns the team about them.

Although there are a total of 28 rules and practices of XP, they can be compacted into 12 simple rules, many of which originated some of the most used and

modern Agile components and methodologies, e.g., Test-Driven Development, Pair Programming and Continuous Integration:

1. *user stories (planning)*: a smaller version of use cases. The customer defines as briefly as possible the specification of the new application regarding features, value, and priority. The stories serve as a base to cost and effort estimation of the project;
2. *small releases (building blocks)*: each newly added requirements will be instantly incorporated, and the system is re-released;
3. *metaphor (standardized naming schemes)*: the developers adhere to standards on names, e.g., class and method names;
4. *collective ownership*: there is not any individual property of the code. All the code can be reviewed and updated by everyone;
5. *coding standard*: developers also adhere to styles and formats of coding to favor the compatibility between team members;
6. *simple design*: the most straightforward implementation of solutions is the most welcome if it meets all the required functionalities;
7. *refactoring*: the application should be continually adjusted and improved by all the team members;
8. *Test-Driven Development*: every small release must pass tests before becoming a new release;
9. *Pair Programming*: the programmers work in pairs in front of a single machine. Essentially, all the code is reviewed as it is written;
10. *Continuous Integration*: the code is continuously integrated and released avoiding the work fragmentation, no more than a few hours;
11. *40-h workweek*: no one works more than what the body can handle;
12. *on-site customer*: the customer is an integral part of the project. S/he has to be available at all the times to ensure the right track for the project.

## 4.1 XP in Education

The characteristics of XP also aroused the interest of researchers in education. In particular, Lembo and Vacca proposed an instructional design methodology that exploits XP and project-based learning (Lembo & Vacca, 2012). They found the XP paradigm as particularly well suited for teaching since everything continuously changes in an educational environment. The students are human, and thus their learning response to teaching is never completely predictable, changing with the time and from student to student. The instructional design process is considered as constituted by roles, e.g., the students, teachers, leadership teams, families, each of them performing some activities, e.g., solving problems, personal study, lecturing.

In the same way as the Agile Manifesto in Education (see Table 3), Lembo and Vacca adapted the values of Agile to the educational environment to apply the XP methodology:

1. the collaboration between students and teachers over processes and tools;
2. the collaboration between students, parents, and teachers over educational agreements;
3. exciting activities over instructional design documentation;
4. the design, problem-solving and task performing over notions and knowledge;
5. responding to feedback over following plans.

The highest priority is to satisfy the students and their parents through the continuous production of real projects, and the achievement of results. A collaboration between teachers, students, and parents takes place during each iteration of the project, preferring face-to-face communication. The educational projects must be designed to solve complex real-life problems and be of a short duration, generating at least the level of expected knowledge, skills, and capabilities. Thus, a project must require critical activities, analysis, synthesis, problem-solving to be applied individually or cooperatively and collaboratively. The project proposals are presented in the form of stories and shared with students and parents, who represent the stakeholders.

The literature presents some works regarding the teaching of XP methodology in software engineering courses (Melnik & Maurer, 2003, 2005; Stapel, Lübke, & Knauss, 2008). Stapel et al. presented the experiences and best practices gained in three different XP labs conducted during three years of a software engineering course (Stapel et al., 2008). They organized the labs to practically experience most of the XP practices. Besides being entertaining, XP resulted in a worthwhile experience to improve the overall programming and social skills. Melnik and Maurer conducted a study over five different academic levels of agile practices, in particular using the XP method, showing that all the students accept and like them (Melnik & Maurer, 2005).

Many works involve the practical use of Pair Programming (PP), one of the most influential practices derived from XP (Anderson & Gegg-Harrison, 2012; Lui, Litts, Widman, Walker, & Kafai, 2016; Umapathy & Ritzhaupt, 2017). Other than numerous experiences in computer science, PP was also used to teach other subjects. For instance, Lui et al. reported the findings from an e-textile workshop in which high school students worked in pairs, observing positive results regarding role distribution, and partner communication strategies (Lui et al., 2016). Furthermore, Anderson and Gegg-Harrison explored pair teaching, where the pairing concept is applied to the teachers instead of the students (Anderson & Gegg-Harrison, 2012). Pair teaching increased the interaction between teachers and individual students, improved the quality of course materials, and gave the students two potential role models.

Moreover, Test-Driven Development (TDD) was investigated in the literature. One example is the one involving the use of the Coding Dojo as an environment to teach TDD (Heinonen, Hirvikoski, Luukkainen, & Vihavainen, 2013; Lee, Marepalli, & Yang, 2017; Luz, Neto, & Noronha, 2013). Coding Dojo is a dynamic and collaborative activity where people can practice programming, in particular by applying techniques related to XP practices such as TDD and PP. Even if Coding Dojos are usually themed by programming challenges, the environment is non-competitive,



collaborative, and fun. The participants were able to learn and practice TDD with a relaxed feeling.

## 5 Scrum

Scrum is one of the most employed process frameworks implementing Agile values and principles. It was created in 1993 and presented to the scientific community in 1995 by Schwaber and Sutherland, two of the authors of the Agile Manifesto. Their work was then collected in a public document named “The Scrum Guide”, trying to define the framework officially (Schwaber & Sutherland, 2011). The term was borrowed from a 1986 article by Takeuchi and Nonaka in the context of product development, published in the Harvard Business Review (Takeuchi & Nonaka, 1986). They found an analogy between high-performing, cross-functional teams and the scrum formation used by rugby teams.

In Scrum, the software is developed by following an iterative model used to manage people and complex projects. Figure 4 shows the Scrum lifecycle. It follows a set of roles, responsibilities, and meeting that never change. With Scrum, the products are built in a series of fixed-length iterations, short and on a regular cadence, defined as “sprints”. During these intervals, teams have the time to develop the software and at the end of each sprint, i.e., the “milestone”, the progress is tangible. By using short iterations, the importance of a good estimation and a fast feedback from tests are reinforced.

Everyone in Scrum plays a specific role

- *product owner*: serves as an interface between the development team and its customers. S/he is responsible for checking that the expectations, in the form of a prioritized wish list called “product backlog”, are respected;
- *Scrum master*: a facilitator within the team members. S/he is responsible for ensuring that the Scrum best practices are respected, and the project moves forward;

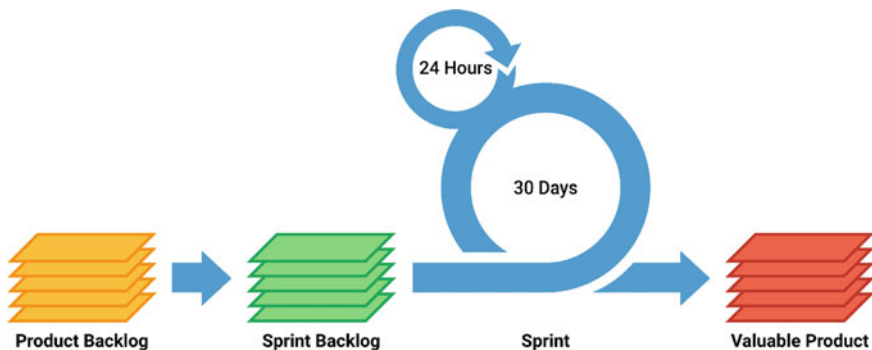


Fig. 4 The Scrum lifecycle

- *Scrum development team*: comprehends the developers that work together to create and test incremental releases of the product.

Scrum defines four events related to a sprint called “ceremonies”:

- *sprint planning*: the team meets and determines what to complete in the coming sprint;
- *daily stand-up (or “daily scrum”)*: a 15-min short meeting for the team to sync on the project progress;
- *sprint demo*: the team shows what has been produced during sprint;
- *sprint retrospective*: a review of the work done and not done, defining the actions to make the next sprint better.

The development team writes down a list of tasks in what is called the “scrum backlog”. These are divided into value-driven “user stories”, a quick way of handling customer requirements without having to create formalized documents.

## 5.1 Scrum in Education

Many researchers worked on ways to adapt Scrum to the educational context. One relevant attempt is given from “eduScrum” (Delhij, van Solingen, & Wijnands, 2015), a guide that translates the Scrum process, roles and responsibilities in pedagogic terms and that can potentially be applied to teach any subject at any education level.

The teacher assumes the role of product owner, who decides what needs to be learned, monitors, processes, and evaluates the students. His/her main goal is delivering the highest value, in terms of both discipline specific learning outcomes and soft skills such as organization, planning, collaboration, and teamwork.

The student team is self-organized and aims at acquiring (delivering) learning results iteratively and incrementally. An eduScrum master, who is chosen by the product owner or the class, acts as a coaching leader and helps the team to perform optimally.

Even the sprints are mapped into the education context. The tasks are considered as time-boxed events with a maximum duration and designed to allow critical transparency and inspection. Thus, the sprints are collections of tasks, coherently organized to achieve the learning goals, and usually have a duration of 2 months or less. The expected ceremonies in eduScrum are

- a planning meeting at the beginning of the sprint, to define team formation, learning goals, and work planning;
- the stand-ups at the beginning of every class, with a duration of 5 min to synchronize activities and make plans for the next meeting;
- a review of the past activities of the last sprint, to display what the members have learned;
- a retrospective to create a plan for improvement and preparing for the future sprint.

As for Scrum used to teach software engineering, the literature reports many experiences where it was successfully employed (Bruegge, Krusche, & Wagner, 2012; Mahnic, 2012; Scharf & Koch, 2013; Smith, Cooper, & Longstreet, 2011; Werner, Arcamone, & Ross, 2012; Zorzo, de Ponte, & Lucreديو, 2013). Scrum is adapted in the context of the development of academic projects, both in undergraduate and graduate courses. The students are organized into small teams and execute the projects following the rules of Scrum. The instructor usually takes the role of product owner and, for each team, one of the members acts as the Scrum master.

Missiroli et al. presented a case study of software development teaching for K-12 education using Scrum (Missiroli, Russo, & Ciancarini, 2017). They designed an experiment in seven classes in different schools, assigning the same software project but realized by two teams of the same class using different methodologies, i.e., the classic waterfall and Scrum. For Scrum, the product owner is not a peer but the teacher. Considering the young age and experience of participants, the authors suggested striking a compromise between the two methodologies, planning, and structure along with creativity and reactivity.

Scrum was also successfully introduced in universities by means of serious games (Fernandes & Sousa, 2010; Lynch et al., 2011; Paasivaara, Heikkilä, Lassenius, & Toivola, 2014; Steghöfer, Burden, Alahyari, & Haneberg, 2017; von Wangenheim, Savi, & Borgatto, 2013). One example is the one related to LEGO-based games (Lynch et al., 2011; Paasivaara et al., 2014; Steghöfer et al., 2017). In the games, student teams learn the Scrum roles, events, and concepts in practice by simulating several development sprints, incrementally planning and building a product of LEGO blocks. Another example is SCRUMIA, which involves a different setup where teams of students are asked to develop different artifacts with the use only of pencil and paper over multiple sprints (von Wangenheim et al., 2013).

Moreover, Scrum was effectively employed as an educational and management method for interdisciplinary education (Cubric, 2013; da Silva Coelho et al., 2014; Gestwicki & McNely, 2016; Ramos et al., 2013). Gestwicki and McNely managed together programmers, artists, and user interface designers to produce 6 different educational game projects (Gestwicki & McNely, 2016). The students came from a variety of degree programs, working in collaboration with one or more faculty mentors and community partners from outside the university.

Finally, Scrum was used to teach other subjects (Duvall et al., 2017; Grimheden, 2013; Mäkiö, Mäkiö-Marusik, & Yablochnikov, 2016; Pinto et al., 2009; Ringert, Rumpe, Schulze, & Wortmann, 2017; Scharff & Verma, 2010). Duvall et al. implemented some Scrum-based classroom management techniques with the aim of getting students to take more responsibility for their learning in a discrete mathematics course at the university (Duvall et al., 2017). The students, divided into teams, enjoyed the self-management and crafting of their learning process. The teams variously chose lecture-based learning, online video-learning, traditional, or interactive online textbook reading. Each of them kept a project management progress board so that the professor could track team progress toward self-selected milestones. Besides the independent work, a few traditional lecture times followed, feeling to the students more like group discussions. Furthermore, Grimheden investigated the use of

Scrum for the teaching of mechatronics, which is defined as a synergic integration of electronics, mechanical engineering, control, and software engineering (Grimheden, 2013). They showed that Scrum enables the students to deliver results faster, more reliably and with higher quality than other methodologies.

## 6 Conclusions

In this chapter, the world of Agile methodologies is explored focusing on their adaptation to the education environment. The philosophy behind Agile and its variations, e.g., eXtreme Programming and Scrum, consisting of values, principles, and best practices, is also maintained in the classroom, where the people factor is particularly important.

The reported experiences suggest that Agile can be effective, especially where active and project-based learning can be applied. Not only can Agile be simulated in software engineering courses, but also it can be applied to teach other subjects. Also Agile tools, e.g., Kanban boards, can be part of the learning process.

Applying Agile methodologies to learning and teaching transforms from knowledge transfer to knowledge generated from rich collaboration and experience. Teachers become facilitators, coaches, and inspirational servant leaders for students that are self-directed learners. The focus is not on rigid plans, rather flexibility is required to take into account students' feedback and their different abilities, interests, difficulties, and experiences, aiming at unlocking their hidden strengths and passions. The emphasis is on delivering the highest value, in terms of both discipline specific learning outcomes and soft skills such as organization, planning, collaboration, and teamwork.

This review of the literature shows that there is a growing interest in studying, but even more, applying Agile learning methodologies to allow students to work together in an energetic, targeted, and effective way. There is also an active effort from researchers in formalizing the agile methodologies in the education context, e.g., eduScrum.

## References

- Abrahamsson, P., Salo, O., Ronkainen, J., & Warsta, J. (2002). *Agile software development methods: Review and analysis*. VTT.
- Ahmad, M. O., Liukkunen, K., & Markkula, J. (2014). Student perceptions and attitudes towards the software factory as a learning environment. In *IEEE Global Engineering Education Conference (EDUCON)* (pp. 422–428).
- Alfonso, M. I., & Botia, A. (2005). An iterative and agile process model for teaching software engineering. In *IEEE International Conference on Software Engineering Education and Training (CSEE&T)* (pp. 9–16).

- Anderson, N., & Gegg-Harrison, T. (2012). Pair<sup>2</sup> learning = pair programming × pair teaching. In *Western Canadian Conference on Computing Education* (pp. 2–6).
- Ashraf, M. A., Shamaail, S., & Rana, Z. A. (2012). Agile model adaptation for E-learning students' final-year project. In *IEEE International Conference on Teaching, Assessment and Learning for Engineering (TALE)* (pp. T1C–18).
- Bacea, I. M., Ciupe, A., & Meza, S. N. (2017). Interactive Kanban—Blending digital and physical resources for collaborative project based learning. In *IEEE International Conference on Advanced Learning Technologies (ICALT)* (pp. 210–211).
- Beck, K. (1999). *Extreme Programming explained: Embrace change*. Addison-Wesley Professional.
- Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., ... Thomas, D. (2001). *Manifesto for agile software development*. Retrieved from <http://agilemanifesto.org>.
- Bruegge, B., Krusche, S., & Wagner, M. (2012). Teaching Tornado: From communication models to releases. In *Educators' Symposium (EduSymp)* (pp. 5–12).
- Bruegge, B., Reiss, M., & Schiller, J. (2009). Agile principles in academic education: A case study. In *International Conference on Information Technology: New Generations* (pp. 1684–1686).
- Cockburn, A. (2004). *Crystal clear: A human-powered methodology for small teams*. Pearson Education.
- Cockburn, A., & Highsmith, J. (2001). Agile software development, the people factor. *Computer*, 34(11), 131–133.
- Cubic, M. (2013). An agile method for teaching agile in business schools. *The International Journal of Management Education*, 11(3), 119–131.
- da Silva Coelho, R. A., da Cunha, A. M., Gomes, A. A., Segeti, E. R., Marques, J. C., Vicente, L. M., ... Mirachi, S. (2014). Developing a CDS with Scrum in an interdisciplinary academic project. In *IEEE/AIAA Digital Avionics Systems Conference (DASC)* (pp. 5D6–1).
- Delhij, A., van Solingen, R., & Wijnands, W. (2015). *The eduScrum Guide* (No. 1.2) (p. 21). Retrieved from [http://eduscrum.nl/en/file/CKFiles/The\\_eduScrum\\_Guide\\_EN\\_1.2.pdf](http://eduscrum.nl/en/file/CKFiles/The_eduScrum_Guide_EN_1.2.pdf).
- Dewi, D. A., & Muniandy, M. (2014). The agility of agile methodology for teaching and learning activities. In *Malaysian Software Engineering Conference (MySEC)* (pp. 255–259).
- Dingsøy, T., Nerur, S., Balijepally, V., & Moe, N. B. (2012). A decade of agile methodologies: Towards explaining agile software development. *Journal of Systems and Software*, 85(6), 1213–1221.
- Duvall, S., Hutchings, D., & Kleckner, M. (2017). Changing perceptions of discrete mathematics through Scrum-based course management practices. *Journal of Computing Sciences in Colleges*, 33(2), 182–189.
- Dybå, T., & Dingsøy, T. (2008). Empirical studies of agile software development: A systematic review. *Information and Software Technology*, 50(9–10), 833–859.
- Fernandes, J. M., & Sousa, S. M. (2010). PlayScrum—A card game to learn the scrum agile method. In *International Conference on Games and Virtual Worlds for Serious Applications* (pp. 52–59).
- Fronza, I., Ioini, N. E., & Corral, L. (2017). Teaching computational thinking using agile software engineering methods: A framework for middle schools. *ACM Transactions on Computing Education*, 17(4), 1–28.
- Gestwicki, P., & McNely, B. (2016). Interdisciplinary projects in the academic studio. *ACM Transactions on Computing Education*, 16(2), 1–24.
- Grimheden, M. E. (2013). Can agile methods enhance mechatronics design education? *Mechatronics*, 23(8), 967–973.
- Haddaway, N. R., Collins, A. M., Coughlin, D., & Kirk, S. (2015). The role of google scholar in evidence reviews and its applicability to grey literature searching. *PLoS ONE*, 10(9), e0138237.
- Hanks, B. (2007). Becoming agile using service learning in the software engineering course. In *Agile Conference (AGILE)* (pp. 121–127).
- Heikkilä, V. T., Paasivaara, M., & Lassenius, C. (2016). Teaching University Students Kanban with a Collaborative Board Game. In *IEEE/ACM International Conference on Software Engineering: Software Engineering Education and Training Track (ICSE-SEET)* (pp. 471–480).

- Heinonen, K., Hirvikoski, K., Luukkainen, M., & Vihavainen, A. (2013). Learning agile software engineering practices using coding dojo. In *ACM SIGITE Conference on Information Technology Education (SIGITE)* (pp. 97–102).
- Kastl, P., Kiesmüller, U., & Romeike, R. (2016). Starting out with projects: Experiences with agile software development in high schools. In *Workshop in Primary and Secondary Computing Education (WiPSCE)* (pp. 60–65).
- Kessler, R., & Dykman, N. (2007). Integrating traditional and agile processes in the classroom. In *ACM Technical Symposium on Computer Science Education (SIGCSE)* (pp. 312–316).
- Kitchenham, B., & Charters, S. (2007). *Guidelines for performing systematic literature reviews in software engineering*. EBSE Technical Report, Keele University. Retrieved from <https://www.cs.auckland.ac.nz/~norsaremah/2007%20Guidelines%20for%20performing%20SLR%20in%20SE%20v2.3.pdf>.
- Ladas, C. (2009). *Scrumban: Essays on Kanban systems for lean software development*. Modus Cooperandi Press.
- Lee, Y., Marepalli, D. B., & Yang, J. (2017). Teaching test-drive development using dojo. *Journal of Computing Sciences in Colleges*, 32(4), 106–112.
- Lembo, D., & Vacca, M. (2012). *Project Based Learning + Agile Instructional Design = EXtreme Programming based Instructional Design Methodology for Collaborative Teaching* (No. 8). Dipartimento di Informatica e Sistemistica “Antonio Ruberti”, Sapienza Università di Roma.
- Lu, B., & DeClue, T. (2011). Teaching agile methodology in a software engineering capstone course. *Journal of Computing Sciences in Colleges*, 26(5), 293–299.
- Lui, D., Litts, B. K., Widman, S., Walker, J. T., & Kafai, Y. B. (2016). Collaborative maker activities in the classroom: Case studies of High School Student pairs’ interactions in designing electronic textiles. In *Annual Conference on Creativity and Fabrication in Education (FabLearn)* (pp. 74–77).
- Luz, R. B. da, Neto, A. G. S. S., & Noronha, R. V. (2013). Teaching TDD, the coding dojo style. In *IEEE International Conference on Advanced Learning Technologies (ICALT)* (pp. 371–375).
- Lynch, T. D., Herold, M., Bolinger, J., Deshpande, S., Bihari, T., Ramanathan, J., & Ramnath, R. (2011). An agile boot camp: Using a LEGO®-based active game to ground agile development principles. In *IEEE Frontiers in Education Conference (FIE)* (pp. F1H–1).
- Mahnic, V. (2012). A capstone course on agile software development using Scrum. *IEEE Transactions on Education*, 55(1), 99–106.
- Mäkiö, J., Mäkiö-Marusik, E., & Yablochnikov, E. (2016). Task-centric holistic agile approach on teaching cyber physical systems engineering. In *Annual Conference of the IEEE Industrial Electronics Society (IECON)* (pp. 6608–6614).
- Meerbaum-Salant, O., & Hazzan, O. (2010). An agile constructionist mentoring methodology for software projects in the high school. *ACM Transactions on Computing Education (TOCE)*, 9(4), 21.
- Melnik, G., & Maurer, F. (2003). Introducing agile methods in learning environments: Lessons learned. In *Conference on Extreme Programming and Agile Methods* (pp. 172–184).
- Melnik, G., & Maurer, F. (2005). A cross-program investigation of students’ perceptions of agile methods. In *IEEE/ACM International Conference on Software Engineering (ICSE)* (pp. 481–488).
- Missiroli, M., Russo, D., & Ciancarini, P. (2017). Agile for millennials: A comparative study. In *IEEE/ACM International Workshop on Software Engineering Curricula for Millennials (SECM)* (pp. 47–53).
- Noguera, I., Guerrero-Roldán, A.-E., & Masó, R. (2018). Collaborative agile learning in online environments: Strategies for improving team regulation and project management. *Computers & Education*, 116, 110–129.
- Paasivaara, M., Heikkilä, V., Lassenius, C., & Toivola, T. (2014). Teaching students Scrum using LEGO blocks. In *IEEE/ACM International Conference on Software Engineering: Software Engineering Education and Training Track (ICSE-SEET)* (pp. 382–391).
- Paez, N. M. (2017). A flipped classroom experience teaching software engineering. In *IEEE/ACM International Workshop on Software Engineering Curricula for Millennials (SECM)* (pp. 16–20).

- Palmer, S. R., & Felsing, M. (2001). *A practical guide to feature-driven development*. Pearson Education.
- Pinto, L., Rosa, R., Pacheco, C., Xavier, C., Barreto, R., Lucena, V., ... Maurçcio, C. (2009). On the use of Scrum for the management of practical projects in graduate courses. In *IEEE Frontiers in Education Conference (FIE)* (pp. 1–6).
- Ramos, M. P., Matuck, G. R., Matrigrani, C. F., Mirachi, S., Segeti, E., Leite, M., ... Dias, L. A. V. (2013). Applying interdisciplinarity and agile methods in the development of a smart grids system. In *International Conference on Information Technology: New Generations* (pp. 103–110).
- Rico, D. F., & Sayani, H. H. (2009). Use of agile methods in software engineering education. In *Agile Conference (AGILE)* (pp. 174–179).
- Ringert, J. O., Rumpe, B., Schulze, C., & Wortmann, A. (2017). Teaching agile model-driven engineering for cyber-physical systems. In *IEEE/ACM International Conference on Software Engineering: Software Engineering Education and Training Track (ICSE-SEET)* (pp. 127–136).
- Romeike, R., & Göttel, T. (2012). Agile projects in high school computing education: emphasizing a learners' perspective. In *Workshop in Primary and Secondary Computing Education (WiPSCE)* (pp. 48–57).
- Scharf, A., & Koch, A. (2013). Scrum in a software engineering course: An in-depth praxis report. In *IEEE International Conference on Software Engineering Education and Training (CSEE&T)* (pp. 159–168).
- Scharff, C., & Verma, R. (2010). Scrum to support mobile application development projects in a just-in-time learning context. In *ICSE Workshop on Cooperative and Human Aspects of Software Engineering* (pp. 25–31).
- Schwaber, K., & Beedle, M. (2002). *Agile software development with Scrum*. Upper Saddle River: Prentice Hall.
- Schwaber, K., & Sutherland, J. (2011). *The Scrum guide*. Scrum Alliance.
- Seman, L. O., Hausmann, R., & Bezerra, E. A. (2018). On the students' perceptions of the knowledge formation when submitted to a project-based learning environment using web applications. *Computers & Education*, 117, 16–30.
- Smith, T., Cooper, K. M., & Longstreet, C. S. (2011). Software engineering senior design course: Experiences with agile game development in a capstone project. In *International Workshop on Games and Software Engineering (GAS)* (pp. 9–12).
- Stapel, K., Lübke, D., & Knauss, E. (2008). Best practices in Extreme Programming course design. In *IEEE/ACM International Conference on Software Engineering (ICSE)* (pp. 769–776).
- Stapleton, J. (2003). *DSDM: Business focused development*. Pearson Education.
- Steghöfer, J.-P., Burden, H., Alahyari, H., & Haneberg, D. (2017). No silver brick: Opportunities and limitations of teaching Scrum with lego workshops. *Journal of Systems and Software*, 131, 230–247.
- Stewart, J. C., DeCusatis, C. S., Kidder, K., Massi, J. R., & Anne, K. M. (2009). Evaluating agile principles in active and cooperative learning. In *Student-Faculty Research Day, CSIS, Pace University* (p. B3).
- Sugimori, Y., Kusunoki, K., Cho, F., & Uchikawa, S. (1977). Toyota production system and Kanban system materialization of just-in-time and respect-for-human system. *The International Journal of Production Research*, 15(6), 553–564.
- Takeuchi, H. & Nonaka, I. (1986, January). The new product development game. *Harvard Business Review*. Retrieved from <https://hbr.org/1986/01/the-new-new-product-development-game>.
- Umapathy, K., & Ritzhaupt, A. D. (2017). A meta-analysis of pair-programming in computer programming courses: Implications for educational practice. *ACM Transactions on Computing Education*, 17(4), 1–13.
- Vivian, R., Falkner, K., & Falkner, N. (2013). Analysing computer science students' teamwork role adoption in an online self-organised teamwork activity. In *Koli Calling International Conference on Computing Education Research (Koli Calling)* (pp. 105–114).
- von Wangenheim, C. G., Savi, R., & Borgatto, A. F. (2013). SCRUMIA: An educational game for teaching Scrum in computing courses. *Journal of Systems and Software*, 86(10), 2675–2687.

- Werner, L., Arcamone, D., & Ross, B. (2012). Using Scrum in a quarter-length undergraduate software engineering course. *Journal of Computing Sciences in Colleges*, 27(4), 140–150.
- Wohlin, C. (2014). Guidelines for snowballing in systematic literature studies and a replication in software engineering. In *International Conference on Evaluation and Assessment in Software Engineering (EASE)* (p. 38).
- Zorzo, S. D., de Ponte, L., & Lucrecio, D. (2013). Using Scrum to teach software engineering: A case study. In *IEEE Frontiers in Education Conference (FIE)* (pp. 455–461).



# Practices of Agile Educational Environments: Analysis from the Perspective of the Public, Private, and Third Sectors



Orit Hazzan and Yael Dubinsky

**Abstract** This chapter examines the application of agile teaching practices from a sectorial perspective, analyzing how education takes place in the public sector, the for-profit sector, and the third sector. We show that most professionals occasionally wear an educational hat, and as such can apply agile teaching and learning practices, modified as needed for the specific environment (e.g., academia, industry, or the public sector). We use two frameworks that we introduced in previous research to analyze agile teaching practices as well as their expression in the three sectors. Our findings promote efforts to expand the scope of agile learning beyond conventional (formal or informal) educational systems to other sectors and organizations.

**Keywords** Agile educational environments · Public sector · Private sector · NGOs Third sector · Teaching · Learning · HOT analysis framework  
*MERGE* analysis framework

## 1 Introduction

In this chapter, we examine the application of agile teaching practices from a professional-development perspective using two frameworks we have previously developed: the HOT (human-organizational-technological) framework for scenario analysis (Hazzan & Dubinsky, 2010) and the *MERGE* model for professional development (Hazzan & Lis-Hacohen, 2016). Using these frameworks, we show that agile teaching practices can be applied to a range of roles across the three main sectors of society (the public, private, and third sectors). The two frameworks provide a new layer of abstraction for the examination of agile teaching practices.

---

O. Hazzan (✉)  
Technion – Israel Institute of Technology, Haifa, Israel  
e-mail: [oritha@technion.ac.il](mailto:oritha@technion.ac.il)

Y. Dubinsky  
Ness Israel, Tel Aviv, Israel  
e-mail: [yaeldubinsky3@gmail.com](mailto:yaeldubinsky3@gmail.com)

© Springer Nature Singapore Pte Ltd. 2019  
D. Parsons and K. MacCallum (eds.), *Agile and Lean Concepts for Teaching and Learning*, [https://doi.org/10.1007/978-981-13-2751-3\\_3](https://doi.org/10.1007/978-981-13-2751-3_3)

The chapter is organized as follows. Section 2 draws on our book *Agile Anywhere* (Hazzan & Dubinsky, 2014) to show that agility is relevant in a broad range of contexts and organizations, including all three sectors mentioned above. Sections 3 and 4 outline the HOT and *MERge* frameworks respectively. In Sect. 5, we illustrate the application of agile teaching practices in the three sectors. Sections 6 and 7 conclude the chapter with further insights into agile teaching practices.

## 2 Agile Anywhere

As we show in our recent book *Agile Anywhere* (Hazzan & Dubinsky, 2014), agility is a state of mind that enables one to cope with and navigate through uncertain situations and change processes. As such, agility can be applied in countless situations, including education. Indeed, several research works discuss the agile approach in teaching contexts (e.g., Berry, 2012; Royle & Nikolic, 2016; Chun, 2004; Lang, 2017; McAvoy & Sammon, 2005; Vuokko & Berg, 2007). To cite one example we highlight in *Agile Anywhere*, the Finnish education system—widely considered one of the best in the world—employs many agile practices, including reflective processes, teamwork and diversity.

The agile approach to education can be summarized as comprising 11 agile teaching practices (see Table 1). These practices are drawn from Hazzan and Dubinsky (2006),<sup>1</sup> where they related to the teaching of software development methods. Here, they are simplified and broadened to apply to any domain.

Later in this chapter we show how these agile practices, appropriately modified for different contexts, can enhance the performance of practitioners in educational contexts in the first, second and third sectors. First, however, we outline the two frameworks used for the current analysis, namely the HOT framework and the *MERge* model, both of which we developed in previous work. The HOT framework deals with the organization level and is useful for the analysis of organizational scenarios. The *MERge* model focuses on the individual and can be used, for example, for the analysis of practitioners' professional development.

## 3 HOT—Three Perspectives of Agile Environments

In previous work (Hazzan & Dubinsky, 2008, 2014) we defined the Human-Organizational-Technological (HOT) analysis framework, and used it for the analysis of real-world scenarios in agile environments. In Hazzan and Dubinsky (2008) we

---

<sup>1</sup>Hazzan, O. and Dubinsky, Y. "Teaching Framework for Software Development Methods," ICSE Educator's Track, Proceedings of ICSE (International Conference of Software Engineering), Shanghai, China, pp. 703–706. © 2006 Association for Computing Machinery, Inc. <https://doi.org/10.1145/1134285.1134396>.

**Table 1** Agile teaching practices (based on Hazzan & Dubinsky, 2006)

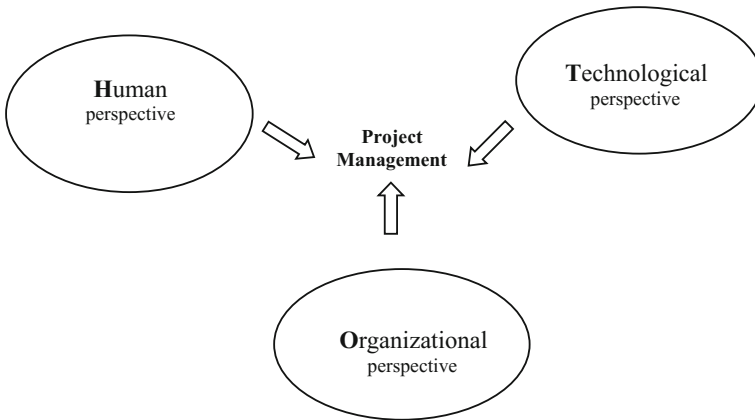
Practice #	Practice description
1	<i>Inspire, don't lecture</i> Agile teaching begins by shifting from a lecture-based model of teaching to an interactive or blended style, with the aim of inspiring students to take an active role in their learning. This is a meta-practice that integrates, and is supported by, many of the practices described below
2	<i>Employ active learning, or learning from experience</i> This practice is derived directly from the previous one. Active learning is based on a recognition that experimentation is essential for effective learning of complex concepts
3	<i>Link explaining to doing</i> This practice, too, follows directly from the previous two. When a new activity or process is introduced, learners should be invited to start applying it as soon as practicable (i.e., once basic relevant knowledge has been introduced). Further explanations can be added and refined while learners perform the activity
4	<i>Elicit reflection</i> The importance of reflective processes has been recognized since Schön's classic <i>The Reflective Practitioner</i> (Schön, 1983, 1987). As in other realms, learning is strengthened when learners are encouraged to reflect on their own learning processes, whether in terms of cognitive processes, feelings, work habits, social interactions, or technical issues related to the project at hand (Hazzan, 2002; Hazzan and Tomayko, 2003). In addition, learners can be asked to recall and reflect on situations taken from their past experience (e.g., in other professional situations)
5	<i>Promote communication</i> In many professional realms, success or failure can hinge on the effectiveness or ineffectiveness of communication between the individuals involved. When communication is one of the main ingredients of the learning environment, the idea of knowledge sharing becomes natural. Accordingly, all learning situations should aim at fostering learner–learner as well as learner–teacher communication
6	<i>Establish diverse teams</i> Technical fields such as software development and engineering have long recognized that project development takes place most effectively in teams, with different members bringing different skills and perspectives to the table. Over recent years, teamwork has been increasingly adopted in nontechnical fields as well. In this context, diversity—whether in terms of nationality, gender, minority membership, culture, lifestyle, or world view—is perceived as a source of added value and a powerful management practice (see, for example, Toyota's 21st Century Diversity Strategy <sup>a</sup> and Thomas, 2004). In the same spirit, diversity is encouraged in agile learning contexts

(continued)

**Table 1** (continued)

Practice #	Practice description
7	<p><i>Assign roles to team members</i></p> <p>In conventional teams, each team member is responsible for particular professional tasks. In agile teams, each member can choose in addition a cross-project role that provides a global perspective on the project (in software development, for example, this might be tester or integrator). This assignment of roles helps ensure that all team members take responsibility for the project's progress. Also, because all team members must become familiar with all parts of the project, such role-taking improves knowledge sharing and learning among team members while supporting more effective project development. This is true in learning contexts as well as professional settings</p>
8	<p><i>Manage time</i></p> <p>In contexts such as software development, time management is crucial to project success. A variety of methods are available for effective time management. For example, iteration-based planning is an agile practice whereby the next cycle of a software development project is discussed and planned through a meeting of the entire team at the start of the cycle. Another agile practice that can contribute to effective time management is frequent pauses for reflection. Such reflection can help catch problems before they arise while promoting deeper learning of complex concepts</p>
9	<p><i>Foster awareness of abstraction levels</i></p> <p>Educators and instructors need to be conscious of the abstraction level employed for each stage of each activity performed during the project development or other learning process. They can then guide learners to think in terms of a different level of abstraction as appropriate, based on learners' needs. They can promote even higher level learning by highlighting these shifts between abstraction levels explicitly, and helping learners reflect on the advantages and disadvantages of such moves</p>
10	<p><i>Use metaphors and analogies (i.e., other concept-worlds)</i></p> <p>Metaphors and analogies help us apprehend complex concepts by drawing on language normally applied to experiences in other realms of life. People use metaphors naturally in all sorts of contexts, including education (Lakoff and Johnson, 1980)</p> <p>There are many ways to bring metaphors and analogies into the learning environment. For example, the educator might say: "Can you suggest another concept-world that could help us understand this unclear issue?" In most cases, learners will be inspired to produce a varied collection of concept-worlds that clarify different aspects of the topic being discussed</p>
11	<p><i>Emphasize the connections between the topic being taught and the larger context in which it operates</i></p> <p>This practice closes the circle opened with practice 1—"inspire, don't lecture." If there is a recipe for inspiring learners, one essential ingredient is linking the specific concept or activity being taught to the larger context in which it will be applied. For example, in software or engineering settings, this can be done by presenting learners with specific problems faced by practitioners in the domain-world, and illustrating how the material being taught can help overcome them. Learners can then see the material as something that has emerged in answer to real-world needs or broader questions</p>

<sup>a</sup>Toyota's 21st Century Diversity Strategy: <http://www.toyota.com/about/diversity/21stcenturyplan.pdf>



**Fig. 1** The HOT analysis framework

focus on software engineering environments, and in Hazzan and Dubinsky (2014), we extend this analysis for project management in general. In this chapter, the HOT analysis framework is used for the analysis of agile teaching practices in each sector.

The HOT framework is based on three main perspectives:

- The **H**uman perspective deals with cognitive and social factors, and relates to interpersonal processes (teammates, customers, management).
- The **O**rganizational perspective deals with managerial and cultural factors, and relates to the workspace and to issues which are beyond the team's scope.
- The **T**echnological perspective deals with practical and technical factors, and relates to how-to issues on a different level of abstraction.<sup>2</sup>

Figure 1 provides a schematic view of the HOT analysis framework.

A brief analysis of the agile teaching practices presented above (Table 1) shows that they can be easily mapped onto the HOT framework (Table 2). For the scenario analysis presented later in this chapter, we denote each practice by its H-O-T hallmark and its original number in Table 1.

## 4 MERge—Management, Education, Research

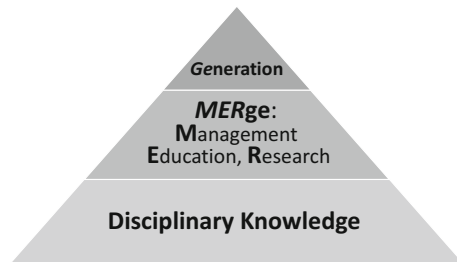
Hazzan and Lis-Hacohen (2016) present the *MERge* model for professional development in industry, academia, and the first sector. The *MERge* model draws on the concept of the meta-profession—namely, sets of skills that extend beyond a specific discipline or area of professional expertise, and that can be expressed meaningfully only after one has first gained disciplinary or professional knowledge (Hazzan &

<sup>2</sup>© Springer: Hazzan, O. and Dubinsky, Y. (2014). *Agile Anywhere—Essays on Agile Projects and Beyond*, Series: SpringerBriefs in Computer Science. <https://doi.org/10.1007/978-3-319-10157-6>.

**Table 2** Mapping of the agile teaching practices according to the *HOT* framework

Human perspective	Organizational perspective	Technological perspective
Inspire, don't lecture (H1)	Promote communication (O5)	Employ active learning (T2)
Link explaining to doing (H3)	Establish diverse teams (O6)	Emphasize the connections between the topic being taught and the larger context (T11)
Elicit reflection (H4)	Assign roles to team members (O7)	
Foster awareness of abstraction levels (H9)	Manage time (O8)	
Use metaphors and analogies (H10)		

**Fig. 2** *MERge* as a three-layer model for professional development



Lis-Hacohen, 2016, p. 3). For example, a software developer may be highly skilled in her discipline, but in order to succeed in a professional environment and fulfill her organizational role she may need to Manage a development team, to Educate graduate students or new hires, or to Research new algorithms or software tools.

As just suggested, the three meta-professions are **Management**, **Education**, and **Research**. The **Management** component includes skills such as time management (on the individual and team level), teamwork, cooperation, communication with customers, and organizational culture. The **Education** component includes skills related to learning, teaching, presentation, team learning, diversity, and many more. The **Research** element includes understanding and using research models, data collection and analysis methods, and drawing reliable conclusions.<sup>3</sup>

The full *MERge* model has three layers. The base, or foundation, is the individual's disciplinary knowledge. The second layer includes the three meta-professions. The third layer—*Generation*—involves integrating the first two layers so as to successfully perform one's professional or organizational role. The full model is presented in Fig. 2.

<sup>3</sup>© Springer: Hazzan, O. and Lis-Hacohen, R. (2016). *The MERge Model for Business Development: The Amalgamation of Management, Education and Research*, SpringerBriefs in Business. <http://www.springer.com/us/book/9783319302249>.

**Table 3** *MERge* in the context of the three sectors—focusing on education

	1 <sup>st</sup> Sector	2 <sup>nd</sup> Sector	3 <sup>rd</sup> Sector
Management			
Education			
Research			

Applying the full *MERge* model to the three sectors produces a 3 × 3 matrix (see Table 3). Since this chapter is about agile teaching and learning, we will focus on the Education meta-profession (shaded in the table).

Below, we employ illustrative field scenarios to show how the *MERge* model (specifically, its education component) is implemented in the three sectors. We analyze these scenarios using the HOT analysis framework, showing the use and impact of the three groups of educational practices (human, organizational, and technological; Table 2) in each sector.

## 5 Application of Agile Teaching Practices in the Three Sectors

Organizations in the three sectors have different goals that affect their vision, activities, and culture. Nevertheless, all kinds of organizations—whether for-profit or non-profit, governmental or NGOs—can benefit from using agile practices in general, and agile teaching practices in particular. This is because the competitive environment in which all kinds of organizations operate means they must be able to quickly and efficiently adjust to changes, either to fulfill their commitment to taxpayers (in the case of the first sector), or merely to survive.

This perspective, which expands the application of (agile) education to all sectors, is one of the main contributions of this chapter.

In this section, we illustrate the application of agile practices in the three sectors using illustrative field scenarios, based on three practitioners whose roles include varying degrees of educational responsibilities. We chose these roles in order to emphasize the suitability of agile teaching practices for any role in any sector, as suggested by the *MERge* model, under which any professional may need to wear an educational hat and therefore needs educational skills.

In the first sector, we examine the case of a school principal. In the second sector, the for-profit sector, we focus on the role of a team leader in a product-based organization. In the third sector, we explore the case of a faculty member at a public university (see Table 4). As we shall illustrate, these roles are multifaceted and their fulfillment can be improved by the application of agile teaching practices.

**Table 4** Roles examined in the present chapter

	1 <sup>st</sup> sector	2 <sup>nd</sup> sector	3 <sup>rd</sup> sector
Management			
Education	School principal	Team leader	Faculty member
Research			

Note that the three role-holders described below are not meant to represent specific figures. Rather, the descriptions are hypothetical and based on our comprehensive familiarity with many practitioners in each sector.

### 5.1 *First Sector: School Principal*

The first (public) sector comprises governmental organizations and local authorities, as well as all organizations which are under their responsibility. This includes government offices, local services provided by local authorities and municipalities, and public schools and hospitals.

Tom is a school principal. He knows that his school does very well, a fact that he ascribes partly to teachers’ professionalism and commitment to their work, and partly to structural features, like the amount of time allocated to teachers for enhancing their professional development. Yet he is also aware that while children, parents and teachers in his school are satisfied at this stage, schools must evolve on a constant basis to adjust to changes taking place in the world and society.

Tom decides to formulate a vision for the school for the next 5 years, as well as (a) a mechanism for updating this vision as needed, and (b) an ongoing monitoring process to ensure that the school does work according to the defined vision during these years. For this purpose, Tom decides to put together a group of teachers, parents, students, and representatives from the Ministry of Education. He calls a meeting to explain the process, and asks for volunteers from each of these groups to attend. Seventy-five people respond to Tom’s call.

In organizing this rather large group of volunteers into an effective body that can develop the school’s new vision and associated monitoring and updating processes, Tom follows five basic principles and actions, shown in Table 5. The table shows how Tom applies agile teaching practices (either consciously or subconsciously) to facilitate a learning process whose aim is to shape a new vision for his school. The list of principles and actions that Tom applies are mapped according to the agile teaching practices presented in Table 1. The practice labels (O5, H4, T11 and so on) are based on the HOT mapping shown in Table 2.

As can be seen, Tom applies agile teaching practices and uses the education meta-profession even though his role in the described scenario involves management, not teaching. That is, from the perspective of the HOT framework, the principal employs agile teaching practices relevant to human, technological, and organizational aspects



**Table 5** Agile teaching practices as used by a school principal

Principles and actions	Agile teaching practices
The group will be divided into teams comprised of teachers, pupils, parents and representatives from the Ministry of Education. The number and size of the teams will be decided by the group. Each team will choose a team leader and decide on its task	Promote communication (O5) Establish diverse teams (O6) Assign roles to team members (O7)
The group will meet every 2 weeks. At these meetings, the teams will present their activity over the last 2 weeks, with a focus on problems faced, what was learned, and their planned focus for the next 2 weeks	Manage time (O8) Elicit reflection (H4)
A collective retrospective session will take place each month to examine what has been achieved so far (Kerth, 2001)	Elicit reflection (H4)
Emphasis will be placed on the fitness of the vision to ongoing social and economic changes affecting society at large	Emphasize the connections between the topic being taught and the larger context (T11)
Progress will be evaluated through quantitative and qualitative measures at the individual, team and school levels. A simple mechanism will be developed for data gathering	Foster awareness of abstraction levels (H9)

of the process. From the perspective of the *MERge* model, the scenario shows how educational skills can help managers lead organizational process.

## 5.2 Second Sector: Team Leader

Private, for-profit organizations comprise the second sector. This sector includes banks, firms, shops, and private services, such as private health organizations, insurance companies, and all other for-profit businesses.

Sharon is a team leader in a large firm in the telecommunications industry. Sharon’s team is responsible for the development of a new mobile application designed to let people use their mobile devices to improve their participation experience during a meeting. For example, features in the app will enable access to information needed for the meeting, automatically record the protocol, track previous action items, and document decisions. The application will also record the involvement of each user in the development process on three levels: participation, action items, and decision-making.

Sharon decides to use a developed version of the app for the development process itself. Table 6 shows examples of the profiles calculated for Sharon’s team during a specific iteration.

In a retrospective session at the end of the iteration, Sharon and her team analyze the team members’ average with respect to each measure. Through this retrospective, the team come up with three new features to suggest for consideration (Schwaber, 1997; Oomen, De Waal, Albertin, & Ravesteyn, 2017). Specifically, they suggest the

**Table 6** Profiles for the development of the new mobile application

	Team members—average (%)	Team leader (%)	Project manager (%)
Participation level	32	45	63
Action items level	60	20	2
Decision-making level	8	35	35
Total	100	100	100

**Table 7** Agile teaching practices as used by a team leader

Principles and actions	Agile teaching practices
By asking her team to use the application being developed (the ‘eat your own dog food’ approach), Sharon shows awareness of different abstraction levels. She also inspires her team with an incentive to improve the product, while encouraging her team to learn through experience and experimentation	Foster awareness of abstraction levels (H9) Inspire, don’t lecture (H1) Employ active learning (T2)
Through retrospective sessions, Sharon promotes communication among the team members and fosters reflection on their experiences	Promote communication (O5) Elicit reflection (H4)
Sharon works with the product owner to prioritize features for development and shorten the time to market. The product owner is a role performed by a project member, who serves as a proxy customer (For more on the product owner’s role, see Sect. 6.)	Assign roles to team members (O7)
The product owner’s role includes representing end users and following plans being developed by the firm’s main competitors. Thus, the product owner’s involvement links the project to the broader context	Emphasize the connections between the topic being taught and the larger context (T11)

app should allow users to plan their own level of involvement and then compare their actual level to the planned level, to time-box planned action items and to connect planned action items to a specific story in the iteration management tool used by the team.

Sharon then meets with product owner (the firm employee responsible for planning what features will be in the released product; see Sect. 6). The product owner, David, agrees that the first feature is important; indeed, he says he has heard that the firm’s main competitor has announced a similar tool. He promises to start planning this for the next iteration.

Table 7 outlines the agile teaching practices used in the learning process which Sharon leads.

### 5.3 *Third Sector: Faculty Member*

The third sector comprises not-for-profit NGOs that supply services not provided by organizations belonging to the first two sectors. It includes charities, philanthropic bodies, and non-governmental organizations (NGOs). In addition, public universities belong to this sector, forswearing profit in order to secure their academic freedom.

University faculty members are evaluated by three main criteria: research, teaching, and service. However, from their first day, faculty members are also required to manage people, including graduate students, lab engineers, and administrative staff. When faculty members are promoted to the role of a dean or other managerial position, they also manage other faculty members.

Here, we focus on an aspect of the faculty member's role that includes both educational and managerial aspects: supervision of graduate students. Diane, an assistant professor in her university's bioengineering department, is researching a breakthrough in medical device technology based on the idea of biological mimicry. Within this framework, Diane supervises five graduate students—three master's students and two doctoral students. Diane assigns each student responsibility for an aspect of the overall research program that is appropriate for their level of experience. As the research progresses, she helps these students develop the skills and knowledge needed to understand both their own part in the research and how their contribution fits into the overall research plan, as well as how that research program itself contributes to the bioengineering discipline by addressing a real-world problem. She also stays alert to the fact that she needs to convey both technical skills and the theoretical knowledge within which those skills must be deployed if her students are eventually going to do their own independent research.

Diane implements agile practices to support the professional development of her graduate students as well as her own professional development. Table 8 shows how agile teaching practices are manifested in this relationship.

## 6 Sub-practices

The present chapter employs two existing frameworks, the HOT framework (used for analyzing organizational scenarios) and the *MERge* model (used to analyze professional practices at the individual level), to examine the application of agile teaching practices from a sectorial perspective. Specifically, based on the *MERge* model, we argue that professionals in the public, for-profit, and third sectors require meta-professional teaching skills. Then, using the HOT framework, we illustrate how professionals holding a representative role in each sector can apply agile teaching practices of all three types (human, organizational, and technology-related).

While the HOT framework, writ large, applies to professional roles in all sectors, certain roles are also characterized by *sub-practices*—i.e., practices that fall under the rubric of one of the main practices listed in Table 1. In what follows, we pro-

**Table 8** Agile teaching practices as used by a faculty member

Principles and actions	Agile teaching practices
<p>Any research process is a long journey, which includes a great deal of uncertainty and many dead ends. In many stages of research work, it is not even clear if meaningful results will eventually be obtained. This can be difficult for graduate students to understand and accept, particularly when the student is concerned about completing their thesis on time. In guiding students through this process, Diane naturally links explaining to doing, modeling the appropriate response to this natural situation. Meanwhile, her students learn through experience and hands-on activity as an active contributor to a research project. Clearly, roles are also assigned very naturally in this process</p>	<p>Employ active learning (T2) Link explaining to doing (H3) Assign roles to team members (O7)</p>
<p>Another of Diane's jobs is helping ensure her graduate students learn time management skills. These are important not only because students face deadlines as they proceed through the graduate program, but also because of the uncertainty involved in any research process (Katz, 2009). Students may not know when to abandon or change a particular line of research. And if time is not managed properly, a student can find himself/herself without meaningful results and eventually, without a degree</p>	<p>Manage time (O8)</p>
<p>Diane and her students meet frequently to analyze the research data and results obtained thus far, and to think about the next stages of the research. This process naturally involves reflective processes and promotes communication</p>	<p>Elicit reflection (H4) Promote communication (O5)</p>
<p>Another natural part of research work is moving between abstraction levels. Sometimes, when research stalls, examining the data from a higher level of abstraction can help clarify what is going on at a basic level. Meanwhile, writing up the results of research work—including in a thesis or dissertation—requires going from a higher level of abstraction to a lower level. This is not a simple process, since it does not reflect the actual path along which the research has progressed. The more aware the student is of abstraction levels along the way, the easier the writing process will be. Thus, fostering awareness of abstraction levels is a crucial agile teaching practice in this context</p>	<p>Foster awareness of abstraction levels (H9)</p>
<p>Researchers often rely on metaphors and analogies to explain their results, especially when the latter are unexpected or based on a new paradigm. This is true even when conveying results to experts in the field, let alone novices like graduate students. In employing metaphors, Diane not only imparts an understanding of the topic at hand; she also models a skill that is useful in any academic discipline</p>	<p>Use metaphors and analogies (H10)</p>
<p>Finally, Diane needs to impart the norms of the relevant research community. In this way, Diane can inspire students with a sense of being part of a larger research endeavor</p>	<p>Inspire, don't lecture (H1) Emphasize the connections between the topic being taught and the larger context (T11)</p>

vide examples of these sub-practices, one for each of the components of the HOT framework.

*Example 1: Reflection-based processes (HOT)*

Engaging in reflection-based processes is a sub-practice of “Elicit reflection” (H4). The agile approach calls for an iterative reflective process that involves all project stakeholders and that relates to each individual’s personal role as well as group processes.

In academic institutions, reflection is part of teaching and research activities. In Sect. 5.3, we described how research involves uncertainty, dead ends, and the exploration of new directions. Such a nonlinear process requires reflection, which also inspires thinking on different levels of abstraction. Educators should also reflect on their teaching on a periodic basis, even—or perhaps especially—in academic institutions, where faculty members’ main role is to research, and the teaching role may be neglected unintentionally.

Aspects of the reflection-based processes sub-practice include

- Promotion of reflective processes on both the individual and the team level.
- Increasing awareness of transitions between levels of abstraction.
- Examining decision-making processes on a regular basis.

*Example 2: Product owner role (HOT)*

The product owner role is a sub-practice of “Assign roles to team members” (O7). The agile approach distinguishes between a Project Manager (PM) and Product Owner (PO), where the PO is responsible for representing the business perspective on the product to be developed (e.g., what capabilities it should feature), and the PM is charged with delivering the product on time and at a high level of quality. In Sect. 5.2, we mentioned Sharon’s practice of reporting results of her team’s retrospective sessions to the product owner. In industry, where competition is high, budgets are limited, and organizations must sometimes thrive to survive, this practice can be crucial.

Aspects of the product owner sub-practice include

- Ensuring that POs know their roles and tasks. These include ensuring that the minimum viable product is expressed in the plan as early as possible; providing feedback to the project team at the end of each iteration; keeping up to date with clients’ needs; and ensuring that these needs are reflected correctly in the work plan.
- Ensuring that the project team are on board and understand the importance of committing to the PO’s plan.

*Example 3: Strategies to anchor change (HOT)*

Develop strategies to anchor change is a sub-practice derived from “Emphasize the connections between the topic being taught and the larger context” (T11). In the second sector, and particularly in technology firms, this might mean using a lean

iterative process and developing the product in a way that will make it easy to incorporate changes in the future. In Sect. 5.1, we mentioned Tom’s goal of not only developing a vision for his school, but also mechanisms for monitoring the school’s performance and updating the vision as needed. Developing innovative strategies to anchor change may be especially important in the public sector, where budget constraints and bureaucracy often mean that organizations are slow to replace outdated technology.

Aspects of the strategies-to-anchor-change sub-practice include

- Awareness that change may be both expected and unexpected;
- The use of lean implementations and mechanisms that enable coping with unexpected changes;
- Being prepared to advance in small steps and to use suitable test mechanisms; and
- Simplification (that is, refactoring) when complexity increases.

## 7 Conclusion

In this chapter, we expand the concept of education in general and agile education in particular beyond the formal and informal educational systems to other sectors and organizations. We convey two main messages. First, we are all surrounded by educational situations in every aspect of our professional lives, no matter what sector we are part of. Professionals should identify these important situations and exploit their learning potential. Second, we provide a new level of abstraction to analyze the usefulness and effectiveness of agile teaching practices as part of practitioners’ professional development.

## References

- Berry, M. (2012). The case for agile pedagogy. Retrieved April, 2016, from <https://www.theguardian.com/teacher-network/teacher-blog/2012/may/16/agile-pedagogy-computer-programming-learning>.
- Chun, A. H. W. (2004, August). The agile teaching/learning methodology and its e-learning platform. In *International Conference on Web-Based Learning* (pp. 11–18). Berlin, Heidelberg: Springer.
- Hazzan, O. (2002). The reflective practitioner perspective in software engineering education. *The Journal of Systems and Software*, 63(3), 161–171.
- Hazzan, O., & Dubinsky, Y. (2006). Teaching framework for software development methods. In *ICSE Educator’s Track, Proceedings of ICSE (International Conference of Software Engineering)*, Shanghai, China (pp. 703–706). <https://doi.org/10.1145/1134285.1134396>.
- Hazzan, O., & Dubinsky, Y. (2008). *Agile software engineering*. Undergraduate Topics in Computer Science’ (UTiCS) Series. Springer.
- Hazzan, O., & Dubinsky, Y. (2010). A HOT—Human, Organizational and Technological—framework for a Software Engineering course. In *Proceedings of the ACM/IEEE 32nd International Conference of Software Engineering (ICSE 2010)*, Cape Town, South Africa (pp. 559–566).

- Hazzan, O., & Dubinsky, Y. (2014). *Agile anywhere—Essays on agile projects and beyond*. Springer-Briefs in Computer Science.
- Hazzan, O., & Tomayko, J. (2003). The reflective practitioner perspective in eXtreme Programming. In *Proceedings of the XP Agile Universe 2003, New Orleans, Louisiana, USA* (pp. 51–61).
- Hazzan, O., & Lis-Hacohen, R. (2016). *The MERge model for business development: The amalgamation of management, education and research*. SpringerBriefs in Business. Retrieved from <http://www.springer.com/us/book/9783319302249>.
- Katz, R. (2009). *Shorten the time to doctorate: A guide to managing your Ph.D. as a project*. AuthorHouse.
- Kerth, N. (2001). *Project retrospective*. Dorest House Publishing.
- Lakoff, G., & Johnson, M. (1980). *Metaphors we live by*. The University of Chicago Press.
- Lang, G. (2017). Agile learning: Sprinting through the semester. *Information Systems Education Journal*, 15(3), 14.
- McAvoy, J., & Sammon, D. (2005). Agile methodology adoption decisions: An innovative approach to teaching and learning. *Journal of Information Systems Education*, 16(4), 409.
- Oomen, S., De Waal, B., Albertin, A., & Ravesteyn, P. (2017). How can SCRUM be successful? Competences of the SCRUM Product Owner. *AIS Electronic Library (AISeL)*. Retrieved from [http://aisel.aisnet.org/ecis2017\\_rp/9/](http://aisel.aisnet.org/ecis2017_rp/9/).
- Royle, K., & Nikolic, J. (2016). A modern mixture, agency, capability, technology and ‘Scrum’: Agile work practices for learning and teaching in schools. *Journal of Education & Social Policy*, 3(3), 37–47.
- Schön, D. A. (1983). *The reflective practitioner*. Basic Books.
- Schön, D. A. (1987). *Educating the reflective practitioner: Towards a new design for teaching and learning in the profession*. San Francisco: Jossey-Bass.
- Schwaber, K. (1997). *Scrum development process. Business object design and implementation*. London: Springer.
- Thomas, D. (September 2004). Diversity as strategy. *Harvard Business Review*, 98–108.
- Vuokko, R., & Berg, P. (2007). Experimenting with eXtreme teaching method—assessing students’ and teachers’ experiences. *Issues in Informing Science & Information Technology*, 4.

# Kaizen and Education



Peet Wiid

**Abstract** In this chapter the concept and origins of *kaizen* are discussed and the difference between *kaizen* and Lean dissected. Although Lean has been popularised in the Western world since 1989, it has unfortunately been a narrow interpretation of the original Toyota Production System (TPS) with *kaizen* as a cornerstone concept. The purpose of *kaizen* should be very clearly stated and aligned with the strategic direction of the specific educational institution. Strategy must be a reflection of ‘customer value’ as monitored through simplicity, quality, speed, cost, motivation, and growth measurements. Although customer value should always be defined (and continuously refined) from all stakeholders’ perspectives, the primary customer remains the student. The creation of a *kaizen* culture is based on seven principles, values, behaviours, and beliefs embedded in the corporate and individual unconsciousness. This culture of excellence will sustain the use of efficiency methods, tools, and techniques. Continuous Improvement efforts in education have mainly failed during the past century. However, with a *kaizen* approach this can be turned around as proven in all sectors. It will require knowledge, skill, experimenting and learning, inspired by committed *kaizen* leadership. Propagating *kaizen* lighthouses of excellence will go a long way to break down the resistance to change.

**Keywords** Kaizen · Lean · Continuous improvement · Lean education  
Kaizen education · Lean teaching · Process improvement

## 1 What Is *Kaizen* and Lean?

*Kaizen* as an organisational excellence approach originated in a manufacturing environment but its principles and methods have been applied in various environments, albeit that education and other service-orientated sectors have been lagging behind

---

P. Wiid (✉)  
Kaizen Institute, Auckland, New Zealand  
e-mail: [peetwiid1@gmail.com](mailto:peetwiid1@gmail.com); [pwiid@kaizen.com](mailto:pwiid@kaizen.com)

P. Wiid  
Manukau Institute of Technology, Auckland, New Zealand

© Springer Nature Singapore Pte Ltd. 2019  
D. Parsons and K. MacCallum (eds.), *Agile and Lean Concepts for Teaching and Learning*, [https://doi.org/10.1007/978-981-13-2751-3\\_4](https://doi.org/10.1007/978-981-13-2751-3_4)



in its adoption (Emiliani, 2015a). It is also important to note that *kaizen* and Lean are not synonymous. Until circa 2007 Lean was propagated mainly as a process improvement methodology with minimal reference to the broader concept of *kaizen* underpinning the Toyota Production System (TPS) with *kaizen* reduced to continuous improvement activities (Womack, Jones, & Roos, 1990; Krafcik, 1988). In many circles Lean has been used interchangeably with *kaizen* or TPS but increasingly since 2007 scholars in the field of organisational improvement started to understand the vast difference.<sup>1</sup>

## 1.1 Defining Kaizen and Lean

According to the legendary Japanese efficiency expert, Masaaki Imai, ‘*Kaizen* means improvement. Moreover, it means continuing improvement<sup>2</sup> in personal life, home life, social life, and working life’ (1986, p. xx). This implies a holistic approach to pursue excellence (organisational and personal) whereby all people are engaged in improving the organisation every day, in all areas. Improvement is therefore not only the responsibility of a few improvement specialists.

According to Jon Miller (who grew up in Japan) the root meaning of *kaizen* is to change for the better by driving out what is bad or evil (inefficiencies in this context) (Miller, Wroblewski, & Villafuerte, 2014). To become better at almost anything requires the application of self-discipline and sacrifice to eliminate bad habits and to replace them with good behaviours that support high performance. Examples of this approach are long-running successful sports teams and athletes, renowned musicians and singers, innovative and consistent business leaders, or outstanding academics. Due to its strong focus on people and their behaviours, *kaizen* has a moral or ethical underpinning which has not been fully recognised and researched by the Lean fraternity. *Kaizen* thus pursues the eradication of what is ‘evil’ and replaces it with what is regarded as ‘good’.

Lean proponents have seen *kaizen* as activities, usually by frontline staff and middle management to make processes better (Ballé, 2010; Womack et al., 1990). In contrast, Miller et al. argue that *kaizen* is a culture, encompassing all behaviours in all areas of an organisation. They state it concisely: ‘...the common thread [is] – that all types of *kaizen* serve to deliver results and develop people.’ (2014, p. 32). These *kaizen* behavioural patterns can be observed in: (i) daily small, incremental improvement activities by frontline staff and lower-level leaders; (ii) improvement projects; (iii) *kaizen* leadership and strategy deployment; and (iv) formal support and coordination of all *kaizen* activities by Continuous Improvement agents (Kaizen Institute New Zealand, n.d.).

---

<sup>1</sup>In this chapter, the author does not view *kaizen* and Lean as synonymous. However, sometimes in quotations, sources are using these words interchangeably.

<sup>2</sup>*Kaizen* is often translated into English as Continuous Improvement.

*Kaizen* is driven by seven principles (to be discussed later) and these differ from the five Lean principles (as described by James Womack and Daniel Jones in *Lean Thinking*) in that the latter focus on the process only. The five Lean principles are: (i) specify value; (ii) map the value stream; (iii) create flow; (iv) establish pull; (v) pursue perfection (Womack & Jones, 1996). The *kaizen* approach is holistic and a representation of the Toyota Production System as introduced to the world outside of Japan by Masaaki Imai in 1986. It also includes the development of the human element in every process. It also expands to the improvement of broader society.

*Kaizen* is a holistic approach to make everything and everyone better; the workplace, processes, policies, people, the environment, the economy, and humanity. The ideal is that everything and everyone must benefit from improvements; *kaizen* does not cause harm (Emiliani, 2015c). It is a techno-social system whereby processes and people are purposefully and continually improving through scientific problem solving that enables the creation of value for the end customer and all other stakeholders.

Lean, on the other hand, is a manufacturing-orientated Westernised interpretation of the Toyota Production System initially studied by Krafcik (1988) and elaborated on by Womack et al. (1990 and 1996). Although Lean has become popular since it was coined by Krafcik, it has not been able to emulate the successes of *kaizen* as developed by Toyota Motor Corporation as its ‘...focus has long been the near-singular pursuit of productivity and efficiency improvements to lower costs and increase profits, usually culminating in lay-offs’ (Emiliani, 2015c, p. 8). From a *kaizen* perspective an organisation does not become lean by being mean.

To better understand Lean and *kaizen*, it is important to gain insight into the historical development of organisational excellence.

## ***1.2 A Brief Perspective on Recent History***

The way organisations behave has improved progressively since the Industrial Revolution. During the late 1700s Eli Whitney introduced exchangeable, standardised parts for muskets to enable the continuous use of a firearm once a defective part has been replaced. Previously the whole firearm had to be discarded (Mirsky, 1998).

Mass manufacturing emerged during the late nineteenth and early twentieth century, replacing craft production. Henry Ford and Frederick W. Taylor revolutionised mass manufacturing through the establishment of the automotive assembly line before and after WWI (Womack et al., 1990). It is regarded in many circles that Taylor’s ‘Scientific Management’ mirrors Western thinking where the focus is mainly on the process, especially its financial benefits for a few stakeholders. In contrast, the Toyota Production System (TPS) is more holistic, a systems approach, leaning towards an Eastern worldview whereby the group (all stakeholders) must benefit (Shingo, 2007).

Conversely, Emiliani (2015b) contends that Taylor did not propagate a focus on process whereby the workers were disregarded. Taylor stated that Scientific

Management ceases when the system delivers bad outcomes for people. It was unfortunately the misuse of Scientific Management by others that led to the belief that Taylor did not care about the workers. Frank and Lillian Gilbreth added to Taylor's scientific analysis (taking his time-and-motion studies to the next level) with a stronger focus on the needs of the employee (Hellriegel, Jackson, & Slocum, 2002).

Albeit, an important reflection on the contributions of Ford and Taylor to organisational improvement is that 'the-winner-takes-it-all' attitude in an organisation will usually lead to disengagement by the negatively-affected stakeholders. Low morale will often derail efforts to satisfy customer requirements and hasten entropy (deterioration) of the system. Emiliani (2015a) speaks of 'non-zero-sum outcomes' as the target condition; a 'win-win' situation for all stakeholders.

Walter A. Shewhart introduced Statistical Control Methods at Bell Telephone Laboratories in New York during the 1930s. This helped to 'recognise when to act and when to leave a process alone' (Walton, 1986, p. 7), bringing about efficiencies by prioritising process problems through standardised response mechanisms. Dr. W. Edwards Deming extended Shewhart's work during the rebuild of the Japanese economy after WWII. His approach to organisational improvement promoted systems thinking (not point improvements), measuring variation in performance, and understanding human behaviour (Walton, 1986). Deming was recognised in 1960 by Emperor Hirohito for his contribution to the rebuild of the Japanese economy.

Although Deming was not directly involved with Toyota Motor Corporation (TMC), his methodology had a profound influence on the development of the Toyota Production System (TPS) as stated by Dr. Toyoda, former president of Toyota Motor Corporation in 2005: 'As we continued to implement Dr. Deming's teachings, we were able to both raise the level of quality of our products as well as enhance our operations on the corporate level.' (Willis, 2012). Other contributors to the post WWII economic revival in Japan include Joseph M. Juran and Kaoru Ishikawa.

Toyota Motor Corporation has been synonymous with organisational excellence. The father of TMC was Sakichi Toyoda who developed power looms during the late 1800s and early 1900s in an effort to make weaving easier for his mother and the workers (Toyota Global Website, n.d.). This respect for people inspired the 45 patents he registered during his lifetime and has since been one of the two pillars of the Toyota Production System (the other is continuous improvement). In 1907 the very successful Toyoda Loom Company was established. TMC was founded in 1937 after Sakichi's eldest son, Kiichiro, visited Ford Motor Corporation in 1927 and had the vision to manufacture vehicles for the Japanese people. Eiji Toyoda, Kiichiro's cousin, also visited Ford Motor Corporation in 1950, which inspired him to pull TMC from the doldrums after WWII. Together with Taiichi Ohno, they realised that Ford's mass production (which relied on large inventory holding, huge and expensive equipment, and high capital expenditure) would not be viable in Japan and this led to the development of a manufacturing system consuming minimal resources, the Toyota Production System (Womack et al., 1990). Taiichi Ohno and Eiji Toyoda developed the Just-In-Time system over a period of 20 years which resulted in extraordinary success and become the subject of ongoing research since

the 1980s. Much of what is covered in this chapter is based on their initial work as well as the contribution of their colleague, Dr. Shigeo Shingo.

Masaaki Imai was the first person to introduce the TPS philosophy of *kaizen* to the world outside of Japan in 1986 in his award-winning and best-selling book, *Kaizen: The Key to Japan's Competitive Success*. Prior to this he worked closely with Taiichi Ohno and the Toyoda family after he spent five years in the United States from 1957 to study American management practices through the Japan Productivity Centre. On his return to Japan in 1961, he became a management consultant (Imai, 1986) working closely with Taiichi Ohno and numerous businesses across the globe. He published a highly acclaimed sequel, *Gemba Kaizen. The Commonsense, Low-Cost Approach to Management* in 1997.

Imai's book on *kaizen* (1986) inspired James Womack (2016) to study TPS and he then popularised 'Lean' with Daniel Jones and Daniel Roos in 1990 with their book *The Machine That Changed the World* (Womack et al., 1990). They introduced Lean manufacturing methods in the 1990s, mainly focusing on cost-savings through improving processes. Unfortunately, it can be argued that their 'Western' paradigm focused their attention on the *methods* employed in TPS with the *respect-for-people* aspect of Toyota sadly ignored. In many organisations this led to the notion that 'Lean is mean', often culminating in headcount reduction (Emiliani, 2015c). It was only around 2007 that Lean advocates started to realise that TPS is more than a cost-reduction methodology. Their corrective action was to introduce Lean Management which gave more attention to Lean leadership and respectful behaviour. Nonetheless, damage to the Lean methodology was already done in some Western economies, especially in North America (Emiliani, 2017a).

Organisational improvement has been pursued for as long as organisations have existed with numerous adaptations, failures, and gains. However, the Toyota Production System is still regarded as the benchmark of excellence due to its holistic and practical approach. But what is the purpose of the *kaizen* system?

## 1.3 The Purpose of Kaizen

### 1.3.1 Developing a Culture of Excellence

The purpose of *kaizen* is to create a sustainable organisational culture of excellence, focused on creating value for the customer by everybody, everywhere in the organisation through continuously solving problems and reducing waste (inefficiencies). Customer value must be quantified in terms of quality, cost, and delivery (speed) (Imai, 1986, 1997).

However, before problems can be fixed they must first be identified, based on a new corporate and individual mind-set of pursuing a better situation. If people do not see their workplace through the *kaizen* filter, they will not be focused on eradicating inefficiencies and satisfying the needs of stakeholders. Once problems have been identified, frontline people (e.g. teachers, lecturers, and coaches) must be

empowered and supported by their leaders to formally solve these in an innovative manner through the disciplined use of the *kaizen* tools. This is the *Lean Thinking* Womack et al. referred to—new corporate and individual thinking patterns (1996).

Albeit, *thinking* is not enough to improve an organisation. *Kaizen* is truly about changing all aspects of an organisation by supporting the creation of new habits of excellence over time, and sustaining these good habits, but also developing it further. A lengthy description of a *kaizen* culture is given in the award-winning book, *Creating a Kaizen Culture* (Miller et al., 2014). An organisation cannot become extraordinary in the way it creates and delivers its goods, services, and ideas, if it is not rooted in a culture of continuous improvement. This culture implies an inward focus to enable the fulfilment of the needs of the ultimate customer (outward focus).

One of the characteristics of this culture is stated eloquently by Wroblewski (2006): ‘The principles of Lean are trying to put harmony into the workplace. This means harmony between man and machine, management and associates, company and customer, company and supplier, and even between company and society. The *kaizen* principles are helping us develop and promote harmony by removing barriers, rocks, and conflicts that disrupt flow in our business.’

*Kaizen*, then, is a holistic approach to improve an organisation but it extends beyond the buildings, the physical classroom, and the virtual training space. It is finally about making humankind better. The spirit of *kaizen* goes far beyond just saving cost. It is a techno-social system, endeavouring to benefit all stakeholders of which the Toyota Production System is a prime example. In an educational setting this implies that the development of students entails more than transferring subject matter knowledge. Creating value for learners includes practical learning; a purposeful application of knowledge to enhance insight and to improve engagement. This insight should eventually extend to the improvement of families, friendships, cultural groups, the environment, sports teams and broader society.

#### ***1.4 Defining What ‘Change-for-the-Better’ Is in Education***

Education will only improve if we know what to improve and why improvement is required. Senior leaders, and subsequently all staff members, must be able to clearly articulate our strategic objectives. The *kaizen* approach is intertwined with developing this vision, mission, values and strategy; it is not merely a few improvement initiatives. We might change structures, curricula, our teaching methods, technology and training aids, but we might just do more harm than good. Not only could students suffer (and have suffered) but also teachers, parents, academia and establishments, with our future as a society at stake. Change for the better should always be synonymous with the customer value we are trying to create. Education primarily exists because there is a need by the student that should be met.

Establishing what ‘better’ is should be done at various educational levels. At the *macro level*, politicians and bureaucrats should provide stability within the education system. Unfortunately, education often becomes a playground for politicians with

major philosophical changes causing confusion and poor outcomes. Bigham and Ray (2012) reported declining reading performance under students when politically influenced curriculum decisions were made instead of data-driven decisions. In New Zealand, misguided political policy decisions over decades have resulted in poor literacy achievement outcomes regardless of recommendations from experts (Turner, Chapman, Greaney, Prochnow, & Arrow, 2013). Harm is caused to numerous stakeholders when their value propositions are not congruent. A self-centred leadership paradigm (as opposed to the servant-leadership model of *kaizen*) is often the root cause of these symptoms. An OECD report (Organisation For Economic Cooperation and Development, 2010) highlighted the need for alignment of national and local policies to overcome school failure.

Defining a value proposition at a *local institution* seems easier as it is closer to the frontline of education. However, determining and measuring performance, (based on an agreed-upon vision, values, principles and strategic objectives), often fails because of the lack of a *kaizen* mind-set and practice. The improvement opportunity for kindergartens, in-home education, schools, tertiary institutes and the trades, is to define simple and clear objectives and to monitor these in a disciplined way on a daily, weekly and monthly basis. Parents with expertise in this field and local schoolboards can be an excellent resource to assist in this regard. But, which objectives should be pursued?

Education should follow the advice from Shigeo Shingo (TPS expert) on how to determine what 'better' looks like in this sector: make things (i) easier, (ii) better, (iii) faster and (iv) cheaper, in that order (1988). This denotes we must first set goals to make our processes **easier** for educators, students, administration, and other stakeholders. Lecturing, preparing for classes, doing research, working on assignments, and using technology should be simplified; not made more complex. It is usually easier to complicate policies, procedures, processes and tasks, than it is to simplify them. A simple key performance indicator (KPI) to show the complexity of the technology employed is the number of technical calls logged per period. Other KPIs can focus on the percentage of learners using technology; the number of critical processes that have been improved and standardised.

Once work has been simplified, it should be made **better** by improving the quality. Poor quality should never be accepted, created, or passed on. These errors and mistakes are difficult to detect because many educational processes are unseen (as in most service-orientated processes). Quality should be experienced in classroom activities, completion of assignments, providing assignment briefs, setting goals, course documentation, the marking of assignments, and in interpersonal relationships, to name a few examples. Setting particular objectives to improve the quality of the input, transformation processes, outputs and feedback loops (Millar & Theunissen, 2008) can greatly enhance the quality of teaching and learning. Examples of setting quality-goals include; student pass rates; the number of teacher training events; the number of standards reviewed or improved; how often performance against standards are checked; or the number of incidents where reports or other information sharing is erroneous.

**Faster** is about speeding up the time it takes to do the work the (internal and external) stakeholders require from us. Idle time, or waiting, is usually a huge part of a process. Very little time is spent on the value-added activities whereby a product or service (like information or documentation) is transformed into the value the customer is expecting. By eliminating waiting a process can be sped up dramatically. When educational processes are evaluated from a *kaizen* standpoint (value stream mapping is a well-known technique) the value-density of the process can be a rude awakening. Faster processes are not the same as rushed processes: the latter indicates poor quality and often higher cost. *Kaizen* can achieve high quality and fast lead times simultaneously. Targets related to speed can be turnaround times when dealing with complaints or hearings; recording month-end deadline breaches; or how long it should take to mark assignments for a specific course.

Lastly, applying *kaizen* also provides the service or product **cheaper**, or more cost-effectively. This is usually the by-product of the previous three goals of making improvements. Too often Western organisations make cost reduction their main reason for applying Continuous Improvement: reduce cost at any cost. The easy (and often lazy) way is headcount reduction or ‘restructuring’. Chasing short-term strategic objectives, set by short-term senior leaders (who are often in the game for their own gain) is frequently the root cause of this debilitating practice (Walton, 1986). Financial targets must not negate the other goals we pursue—the *balanced scorecard* (Balanced Scorecard Institute, n.d.) is often used to ensure strategic synergy. Typical cost goals are: adherence to budgets, decreasing tuition cost, teacher–student ratio; and student debt.

Process simplification impacts favourably on quality, and both these make the process faster, and the culmination of simplification, higher quality and faster processes is cost reduction. Cutting cost without this deeper understanding is futile and most often results in hardship for the organisation, its people, students and families, the economy, and even society. *Kaizen* aspires to bring no-harm to all stakeholders when setting and monitoring the institution’s goals.

These simple four objectives can serve any educational institution well when embarking on a *kaizen* transformation. However, Shingo’s purposes of making improvements are more *process* orientated. Expanding the purpose of *kaizen* in education, the following *people*-orientated target conditions must also be included in any strategic plan:

**Health and safety** of our people includes more than providing a physically safe environment. The emotional welfare of staff members is just as important, especially in the service industry where many processes and their associated problems are more hidden than in manufacturing, often causing stress and burn-out due to work overload. The academic environment can also provide a breeding ground for bullying, as well as demoralising class and wealth discrimination (Emiliani, 2017b). If an institute is serious about making the workplace better, it should also define and monitor health and safety issues.

Improving **morale** and staff satisfaction must be at the heart of an educational organisation. Delivering quality outcomes is to a large degree dependent on the skills, attitudes, and emotional and social intelligence of staff. *Kaizen* develops all people

in an organisation so they are able to spend more time on value-added activities but also to improve their work. These improved capabilities also improve self-confidence, self-discipline, pride, cooperation and trust (Imai, 1997). A simple tool to determine staff satisfaction is the *Net Promoter Score* with employees answering the following question: ‘How likely is it that you would recommend this university (or school) to a friend or colleague?’ (Net Promoter Score, n.d.). However, very often staff satisfaction surveys do not improve morale; they might have the opposite effect. This can be ascribed to several factors: (i) the sincerity and credibility of senior management—do they really care about and serve their staff?; (ii) infrequent surveys with minimal feedback; (iii) no action after surveys; and (iv) the over-arching culture in the education sector. A better technique to improve staff satisfaction is through *gemba* (frontline) walks by supportive leadership on a very regular basis. These scheduled visits to the classroom are not to micro-manage people but to support people to reach the strategic objectives of the team. Process performance is monitored and corrective action taken by both leaders and teachers to enable continuous improvement. This ‘immediate feedback’ is based on the explicit values of respect for people, care, and trust.

Over and above Shingo’s purposes for Continuous Improvement, an educational organisation must also have **growth** aspirations. This might include a roll increase target or a revenue and funding increase to cater for capital projects and operational expenses. Without realistic growth ambitions a school, college, or university will gradually be overtaken by the effects of entropy. *Kaizen* does not only reduce but also increases. This implies you might have the most efficient processes and the most capable lecturers but minimal students and/or funds to justify the institution’s existence. *Kaizen* works best when inwardly focused process-and-people improvement is balanced with outwardly-orientated growth aspirations. Conversely, pursuing a growth strategy per se without improving processes and developing staff can easily lead to failure as *muda* (Japanese for waste) will also increase if not deliberately targeted.

The purpose of *kaizen* in an educational organisation can be condensed to the following: making teaching and learning processes easier, safer and healthier while improving the quality of everything, making processes faster without being rushed or strained. A by-product of all these actions is usually cost savings although explicit financial objectives should also be pursued. Growth aspirations for an institution ensure the benefits of process improvement and people development are maximised. Albeit, after we have defined all these lofty goals, the acid test is summed up in this report: ‘Only when the data meets the student in the classroom will teachers begin to embrace its relevance’ (Lambert, n.d.). The purpose of having a purpose is to primarily improve the student and teacher.

Now that the purpose of *kaizen* in education has been discussed, the broader principles underpinning *kaizen* will be explored.



## 2 Foundational Principles of *Kaizen*

To practise *kaizen* a team must understand the foundational beliefs, principles, values and habits<sup>3</sup> driving efficiency and effectiveness. The seven *kaizen* principles according to Coimbra (2009) are: (i) create customer value; (ii) eliminate waste; (iii) engage people; (iv) go to *gemba*; (v) manage visually; (vi) process and results; and (vii) pull and flow.

Coimbra states that a paradigm shift is required to create new habits based on these beliefs, principles, and values (ibid). It often requires unlearning the ‘traditional’ ways of both teaching and managing educational institutions based on critical reflection. Understanding the connection between the principles and the improvement tools can prevent inauthentic *kaizen* (and subsequent harm to stakeholders) and therefore supports a sustainable *kaizen* journey (Grabau, 2007).

*Kaizen* practitioners continuously research and improve their understanding of the foundational principles, assumptions, values and habits. It must be ingrained in the unconscious mind by the creation of new neural pathways through regular visitation (Mind Warriors Limited, 2009). The more the principles, beliefs and values are applied, the stronger the new *kaizen* habits will become. The *kaizen* principles are subsequently explored.

### 2.1 Create Customer Value

An educational institute exists because it meets certain needs of a customer; in other words, creating products or services that the customer perceives as being of value. It is, however, important to pinpoint what the value is that the customer requires. According to Emiliani, ‘Quality in higher education remains largely undefined’ (2015b, p. 33). He lists 45 common, unforced errors occurring in teaching processes that devalue a teaching system.

It can be correctly argued that an educational organisation has multiple customers with varying, even conflicting requirements. For example, government priorities and policies might not be aligned with student expectations. Prioritising these wide-ranging requirements can be a minefield. Nonetheless, it should be the aspiration to determine a common and simplified understanding of what *value* is and the alignment of all stakeholder value propositions. This will take time and effort but it is achievable in a *kaizen* environment. If dictated government policy does not address the needs of

---

<sup>3</sup>**Principle:** a fundamental truth or proposition that serves as the foundation for a system of belief or behaviour or for a chain of reasoning (Oxford Dictionary).

**Beliefs:** Something one accepts as true or real; a firmly held opinion (Oxford Dictionary). Also called assumptions.

**Values:** Values are deeply held views of what you find worthy. (Mind Warriors Limited, 2009). Not to be confused with *customer value* (the customer’s requirements).

**Habits:** A settled or regular tendency or practice, especially one that is hard to give up (Oxford Dictionary).

the grassroots institution, the latter should work towards clarifying its own strategy so it can feedback to bureaucracy using data to negotiate better alignment. This will require mutual trust.

Value is not only related to the perception and experience of the student (an external customer). Internal customers (staff members, senior management, administration, lecturers, professors and researchers) must also be taken into consideration. Liker and Meier (2006) stipulates that the starting point in the *kaizen* approach is ‘generating value for the customer, society, and the economy.’ It is not first and foremost about cutting cost. Monetary saving is a natural outcome of creating, producing, and delivering exactly what the customer requires when it is required. For instance, the economic benefits of a well-organised, well-skilled society can be compounding. Not only can it help reduce poverty levels, it can also increase social stability. Determining ‘value’ is often described and quantified in terms of Quality, Cost and Delivery (Imai, 1986).

The **quality** component can be measured as the number, percentages, or cost of, failures, defects, mistakes, rework, incomplete work, complaints, non-compliances, etc. The quality of teaching and learning should be quantified through the setting of appropriate targets and monitoring of performance. However, qualitative observations of behaviour, emotions, and attitudes (of student, staff, and other stakeholders) must also be noted and corrective action taken based on the explicit values of the institution. Quality usually starts with simplifying teaching and learning—not complicating it (Shingo, 1988); not by adding more workload and more steps to a process.

The **cost** aspect can refer to budget adherence, cost of providing a course or service, labour cost, cost centre management, government funding, allocation of funds to various departments, outstanding student fees, space utilisation, productivity and so forth. According to Emiliani (2016) higher education in the United States has been under financial pressure for a long period due to decreased student enrolments, increased operating cost and reduction in government funding. Traditional management style cost-cutting is contrary to the *kaizen* way whereby cash flow improvement is achieved through the meticulous improvement of processes (Kaizen Institute USA, 2018). The cost of providing education can be controlled in innovative ways as reported by the Davis Educational Foundation (2012) inquiry into the rising cost of higher education in New England, USA. Some of their suggestions include: (i) year-round use of the campus; (ii) early identification of students not ‘college-ready’ as the remedial work can be costly; (iii) reduce time to graduation; (iv) and blended learning or on-line courses.

**Delivery** has to do with the timeliness of providing services or information. Monthly reporting deadlines come to mind, time wasted in meetings, inconvenient class times and rosters, working overtime to mark complex assignments, waiting for decisions, time allocated to administrative tasks, etc. A *kaizen* education system will endeavour to minimise time spent on activities not adding value to the customers of the system.

Although ‘not everything that can be counted counts, and not everything that counts can be counted’ (Cameron, 1963), it is important to know if a team is improving or falling into entropy. Therefore, measuring the performance of processes (and

people) is a vital *kaizen* activity. Targets should be aligned with what the customers (all stakeholders) require, however, it is not always easy to determine measures in the beginning of a *kaizen* journey due to the instability of the system. Using plain KPIs to highlight key problems in a pilot area can be a sensible way to start.

A key learning is to reduce the number of targets as too many measures will confuse and demoralise. It is therefore important to develop KPIs that will measure critical success factors (CSF). Chasing the ‘wrong’ targets will create inefficient habits, a waste itself. As the *kaizen* journey continues, the targets themselves must also be enhanced through simplification and by combining various objectives to try to reduce these into a single and simple KPI. This can only occur if education leaders and senior managers deliberately and critically reflect on organisational KPIs and associated goals and strategies.

Customer value is constantly being prevented due to waste in processes as will be examined in the next section.

## 2.2 Remove Inefficiencies or Waste

What is waste? The Japanese refer to it as *muda*—not getting paid for an effort. It is consuming resources without adding any value or benefits to the end customer of the process (Imai, 1986).

*Muda* cannot be identified and removed effectively if a clear understanding of what value is has not been predetermined. Otherwise activities might be removed that are not wasteful, or, time can be wasted on fixing processes that should not exist in the first place. Eradicating *muda* becomes more obvious and effective once value has been clearly defined.

Various types of *muda* can be identified in the workplace. Eliminating these inefficiencies is an easy way to start improvement activities, as Masaaki Imai states in his best-selling book, *Gemba Kaizen* (1997). The classic 7-Wastes can easily be remembered by the acronym, *T-I-M-W-O-O-D*:

**Transportation** entails the unnecessary movement of information or materials in a manufacturing setting. In teaching, the ‘materials’ are the students moving through educational processes while being transformed (like raw material is transformed into a more valuable object during a manufacturing process). This can include their inefficient physical movements between classrooms, campuses and travelling long distances for just a one-hour class per day. A frustrating scenario is when students travel long distances to a campus to find the lecturer is unavailable. The root cause of this *muda* is often a disrespect for people. The waste of transportation can also include unnecessary emotional swings (movements away from equilibrium) due to insecurity, unsafe campuses, bullying, or frustration with the quality of teaching or environment. The transformation of the individual takes longer or might even be impeded.

**Inventory** is the storage of information or materials while it is waiting to be used or to be transformed. It piles up in email inboxes, trays, printers, servers, meeting

minutes, course brochures and even on visual boards. Stock items also refer to storing an excessive quantity of consumables and teaching resources, or running out of stock items required to teach. This waste often leads to the waste of 'defects' as unnecessary or over-produced items are often discarded. The trap is to buy more because items have been discounted by the supplier. It only takes a few items to be discarded to nullify the cost benefits of buying in bulk. Bulk-buying also requires bulk storage that could have been used for more productive activities. Too many stock items also lead to more searching (waiting) by staff.

The storage of unnecessary information in a data system also leads to multiple inefficiencies, for example the difficulty to find the correct template, numerous versions of documents, and complicated folder structures. Naming conventions can greatly assist to standardise information record-keeping and standardised folder structures can reduce searching as well.

Asking the following questions can assist with inventory reduction: do we need to keep this? Why? How many? Where? When? Who is responsible?

**Motion** involves unnecessary human action by the operator in a factory (lecturer, teacher, facilitator or a coach in education) like walking too much, and searching for people, information or materials. Too many keystrokes to access information in a complex folder structure indicates motion waste. Too much movement of people can lead to unsafe practices and injuries. Excessive emotional motion can also devalue the participation of the teacher. Low levels of respect for colleagues' workload and their frustrations often result in the overthinking of issues and the spending of emotional energy on self-preservation and conflict resolution. The lack of care for students will also create negativity and hinder performance. A *kaizen* culture enables an environment where debilitating emotions are minimised.

**Waiting** occurs in most processes and huge gains can be made if waiting times can be reduced. Unnecessary and prolonged meetings are a well-known example of this in education. The root cause of this frustrating practice is usually poor or rushed planning. This often results in rework (a quality issue) when another meeting has to be convened or discussion points have to be revisited. Drawn-out decision-making keeps staff and their teams busy while value-added work moves lower down the priority list. Waiting for decisions by leaders or managers also prevents staff from doing better work. Submitting and publishing assignment results late are also not adding value to lecturer or student. These delays are more often the result of cumbersome processes; not uncooperative staff members.

**Over-production** is producing too much information or material before it is required and then it waits while it is stored somewhere, running the risk of turning into a defect. Teachers must be tuned-into their students to identify when they are overloaded with too much work and either reduce assignments or provide timely support to help them cope. In *kaizen* less is often more. Too much (ineffective and inefficient) teaching will result in defective knowledge assimilation which reduces the quality of learning and living. The approach to overload students might also lead to poor work habits in these future employees, managers, and leaders. Nonetheless, teaching should also not pamper students as disciplined learning and good routines will empower students to better manage the challenges of later life.

The Princeton Review provides some useful tips on avoiding over-production at school. It includes studying more often in shorter sessions instead of long, tedious hours, less often. This requires good planning and a set routine. Developing open, trusting communication with teachers and parents to obtain support when a student is struggling is vital. Celebrating successes is also crucial to keep motivational levels high (The Princeton Review, n.d.). These tips are all associated with a *kaizen* approach.

**Over-processing** happens when a process (work) is too complex or difficult and in need of simplification. Marking assignments and performing all the related administrative tasks is usually a real tester for teachers and lecturers. Complaints about unnecessary administrative work in schools and universities have driven numerous excellent teachers from this future-creating vocation (Lambert, n.d.; Allpress, 2018). This waste of over-processing is often leading to the defect of teachers leaving the sector or moving to other schools. The attrition cost in US education is annually between \$1 billion and \$2.2 billion (Alliance for Excellent Education, n.d.).

**Defects** in education are numerous: incorrect data, endless reports, omissions in administrative documents, and justified student and parent complaints. Emiliani (2015a) refers to 45 common, unforced errors occurring in teaching processes that devalue the product delivered to students. Some of these are teachers who cannot teach, lecturers who cannot explain the course content with clarity, go too fast, read from books and slides, do not use real-world examples, come to class unprepared, do not keep to class times, ignore student feedback, cancel classes and speak to students in a condescending way. The obvious defect is a student failing to develop holistically and to underperform. Schools not dealing decisively with bullying are also a defect (Green, Harcourt, Mattioni, & Prior, 2013).

### 2.2.1 The 3Ms

*Muda* forms part of a triad that also includes *mura* (variability, irregularity, unevenness) and *muri* (strain on people, processes and equipment or other technology). This triad is called the 3Ms and they are intertwined. Unevenness in processes leads to strain which results in *muda*, for example; a student not studying regularly in short sessions usually ‘crams’ information just before an exam or assignment in one long session (large batch of information) which often leads to strain (stress, anxiety, lack of sleep). This overburdening can result in various *muda*: (i) defects (poor memory and low retrieval of information, even failing an assignment); (ii) other academic or personal activities waiting; (iii) over-processing of the learning material (re-reading to gain insight); or (iv) slow transportation of knowledge or skills (inventory) to the long-term memory functions of the brain. Education can become better by ‘reducing unnecessary, unreasonable and uneven activities’ (1973 Toyota Production System Manual, p. 2).

Other forms of waste can also be found, for example, marking large batches of assignments at end-of-term instead of getting closer to the ideal of single-piece flow. This can imply shorter assessments more often through ‘machine evaluation’

as Emiliani (2016) implemented for 45% of his course assessments at Central Connecticut State University. ‘Multi-tasking’ can also be very ineffective and inefficient which implies that students (and facilitators) involved in too many courses simultaneously can impede the quality of learning and teaching. The use of mobile devices for social communication during tuition and individual studies can also reduce the quality of learning as attention is continuously diverted (Weimer, 2018). Other inefficiencies in a service environment include work-time losses like absenteeism, and employee underperformance due to low morale.

The 3Ms should be eliminated through daily, small, incremental *kaizen* by all staff members (Imai, 1986). Waste can also be minimised through project-based improvements like Value Stream Design whereby the current state of a process is analysed and then vastly improved to incorporate pull and flow principles. It is also called: ‘Learning to See’ (Rother & Shook, 1998). Daily *kaizen* should once again follow such a project to ensure the improved ways are followed and further enhanced. These projects can deliver break-through results and can be applied at various process levels.

To conclude this section on eliminating waste a word from the efficiency expert, Deming. His rule of thumb is that about 94% of all problems in education will be due to the system (the responsibility of senior leaders) and only approximately 6% can be attributed to employees (Deming, 1986). Senior pedagogical leaders and managers must develop the *kaizen* habits of actively supporting people at the coalface to solve these systemic issues; not blaming them as they struggle against the system in the organisation.

### **2.3 Engage and Develop People**

Engaging people is underpinned by a deep-seated respect for people (and society in general). The Toyota Production System has been an outstanding example of an effective and efficient organisation due to their balanced, holistic approach: people and process should equally and simultaneously become better. Imai (1997) states that engaging people requires ‘everybody, every day, everywhere’ doing *kaizen* for the betterment of all.

An educational organisation must be a ‘learning enterprise’ as Imai further stipulates (ibid.). This does not imply the process of teaching students; it is about staff development. Administrative people, management and frontline educators, are continuously thinking about the systemic problems and process challenges they are facing daily. They reflect regularly on what happened (the good and the bad) and collectively search for solutions to embody a better way of meeting student and other stakeholders’ needs. Mark Graban says ‘Lean [*kaizen* in this context] is a thinking process more than a simple to-do list of tools to implement.’ (2009). It is pointless if we try and ‘fix’ processes without developing the ‘fixing’ skills of teaching staff and administrative personnel.

Engaging people is also about improving staff morale through genuine support to all people. Richard Branson and other high-profile business leaders are convinced that high staff satisfaction underlies customer satisfaction. As Zappos stated it cleverly in an advertising campaign: ‘Happy People Making People Happy’ (Mullenlowe U.S., 2010).

A few aspects related to developing staff should be considered:

- Understanding and practising respect for people, whether they are the (internal or external) customers of the work that is done, or whether they are suppliers of information and materials.
- Respect for people (and broader humankind) must be practised and promoted by the senior leaders at the educational institution. This might require the development of new behavioural patterns for some due to deeply embedded poor practices and systems. However, *kaizen* enthusiasts should not be discouraged by the challenges ahead. Their focus should be on the long-term vision of empowering our societies with knowledge and skill in a relevant, effective and efficient way. We might not even see some of the results in our lifetime but our *kaizen* efforts should create a legacy that future generations can build on.
- Engage faculty and administrative people to remove inefficiencies and to improve quality. In other words, empowerment them to identify waste and to remove it.
- Continuous skill development in the daily processes of teaching, research and administration. Nonetheless, employees should also be trained in the use of the continuous improvement approach and subsequent techniques.
- Deep reflection on practices with team members to find better ways, to standardise the better ways, and to further enhance them.
- Improving staff morale by connecting genuinely and sincerely with people, especially as a leader (at any level in the organisation).
- Celebrate success and reward staff for improvements made and targets achieved. This does not have to be financial. The pride and emotional connection with the workplace can be more powerful than monetary rewards.
- Servant-leadership is required to engage people and to develop them. Coaching staff cannot occur when egos and selfish motives get in the way.
- High staff turnover is one of the most inefficient and devastating results in an organisation: ‘As a rule of thumb, the cost of employee turnover is estimated to be one to three times the departing employee’s annual salary, depending on factors such as the seniority of the position, and how quickly a replacement can be found and trained.’ (Cole, 2001). Organisational efficiency is much more than just measuring process performance. It has a lot to do with the quality of social relationships, the emotions generated within the team, and the lived values of each individual (Miller et al., 2014). Deming asserts that ‘A system that fosters an atmosphere of receptivity and recognition is far preferable to one that measures people by the numbers they turn out.’ (Walton, 1986).

Engaging and developing people happens at the coalface; the theme of the fourth principle of *kaizen*.

## 2.4 Focus on Gemba

The *gemba* is the frontline of the organisation; where value is created for the customer but also where *muda*, *mura* and *muri* persist and where it must be eliminated with the active support of senior leaders (Imai, 1997). Leaders at all levels need to be strongly connected at the coalface of education. Leaders need to derive their decisions and strategic objectives by what is happening where students and educators connect. The opposite of *kaizen* management is managing through reports and endless meetings, behind a comfortable desk, and managing on the internet or in the cloud.

With *kaizen*, problems are made visible at the *gemba* through visual management. When leaders step into the classroom regularly they support and develop teachers to resolve the apparent issues; not to spy, criticise or demean them. A leader's standard work must deliberately be designed to provide optimum support to frontline staff. The lower a leader is in the organisational chart, the more frequent the coalface interfaces will be.

Immediate feedback mechanisms must be designed to ensure rapid exposure of problems, challenges, issues and improvement opportunities. These problems can be fertile ground to enhance people's problem solving abilities and to strengthen their *kaizen* skills and confidence through the coaching of a senior leader. Immediate feedback can be given through the use of visual boards in smaller teams (or departments) whereby daily and weekly performance can be observed and corrective action taken. The early staff room gathering in the morning before school or after lunch is an ideal time for this.

Walking through the *gemba* must be deliberate, well-designed (standardised) visits not only to the coalface but also to the administrative teams. The *gemba* walk is much more than Management By Walking Around (MBWA). Standards must be checked. These include checking if people are following: (i) work instructions, (ii) processes, (iii) achieving student targets, and (iv) perform against teacher-set targets. This checking is in the form of respectful support.

The focus by management (and all other support areas) on the frontline is to ultimately support the teaching staff to solve the problems they and their customers, the students, are facing. It is, however, important to prioritise the problems that are having the biggest impact on creating performance gaps. Once a problem has been identified, defined, and prioritised, root causes should be uncovered and addressed through ideation and creative solutions. As part of the Scientific Method, the results of the implemented solution(s) must always be verified to establish if improvement has been made. If so, standardisation should follow. If the situation did not change, further root cause analyses should be done, or alternative solutions investigated. It is also recommended to solve one problem at a time but do it thoroughly (no multi-tasking) and to achieve the targeted outcomes.

"The greatest sign of strength is when an individual can openly identify things that did not go right, along with 'countermeasures' to prevent these things from happening again." (Liker & Meier, 2006, p. 14).



## 2.5 *Manage Visually*

Visual management is the word used in *kaizen* to describe the management style. It makes information and activities visual so problems (deviations from standards) become obvious. Imai puts it as follows: ‘This is visual management: making abnormalities visible to all employees – managers, supervisors, and workers – so that corrective action can be taken.’ (1997, p. 96). It can even be added that abnormalities and problems can be made visible to students and other stakeholders as well to enable their participation in improvement activities.

Managing classrooms, student areas, and the back offices in a visual manner creates interest and engagement from colleagues, senior management, and students. More improvement ideas can be generated if more eyes are looking at the problems a team is facing.

Visual management creates a disciplined approach to improving the teaching environment as problems can be seen and it creates an urgency to solve it. Without this urgency (about solving the right problems at the right time) improvement efforts will always have a low priority. The content of all team boards throughout the organisation should be aligned, culminating in an overarching, high-level board at senior management level depicting the organisational performance and its people development; a line of sight throughout the organisation.

Visual management can also be seen in the use of videos, presentation slides (minimal use), photographs, graphs, an idea system in the office (and even the class), displaying visual class standards during lectures, or a need-to-know area with important information to save time in class.

An indispensable form of visual management is 5S (better workplace organisation) (Imai, 1997). It is based on five words starting with ‘S’:

1S—**Sort** out an area (physical or digital) by eliminating all items not required to carry out the work.

2S—**Set**-in-order to help locate materials, tools, software, files, folders, data and other information, easily and always in the same place. This can be depicted through photos of the layout, labels, demarcated areas, and naming conventions for files, and standardised folder structures.

3S—**Shine** or sweep the area regularly to ensure compliance and a work environment conducive to high performance. It helps to detect unnecessary files or folders within folders; these should either be deleted or archived. A regular sweep with the eyes when entering a classroom, office, virtual space, or the lunchroom is a habit that can prevent the reoccurrence of waste.

4S—**Standardise** the improvements made during the first 3Ss. Create visual standards to show the target condition in a specific area, preferably by engaging the people using these standards. They usually know the processes better and can be more efficient and effective in creating these standards. The key to effective standardisation is simplification. A picture paints a thousand words is truly applicable here. These visual standards are intuitive and easy to understand and make the deviation from

the standard obvious. Nonetheless, these standards must be improved by the people using them. This can only be achieved with a motivated and inspired workforce.

5S—**Sustain** the standards through regular checks or audits to ensure people are adhering to the better ways that has been developed. Display the results of these checks with clear actions on how to get the area or process back to the standard. This can also display new ideas on improving the existing standard.

Applying 5S must be lead and supported by senior management to ensure the discipline is upheld in all areas. A good area to start with 5S is the staff room of the institution as it is a neutral area where staff can learn-by-doing *kaizen*. It can also become a benchmark and training ground for people on giving the first steps in organisational improvement.

### 2.5.1 Immediate Feedback and Visual Management

Visual control must be of such a nature that a problem can be seen immediately. At Toyota Motor Corporation they have developed the *andon*, a visual and/or audible signal to attract the attention of the supervisor the moment a problem occurs on the assembly line. In similar fashion the next layer of management in education should know as soon as possible when a problem occurred so that root cause analysis with the appropriate people can be done. This urgency to solve problems has the potential to eradicate reoccurring problems through standardisation once a solution has been implemented.

Visual management is a key component of a *kaizen* organisation, but, what should be managed visually? The next section explores the sixth principle of *kaizen*.

## 2.6 Process and Results

Excellent results in education are consistently achieved if the (i) teaching, (ii) learning, and (iii) administrative (supporting teaching) processes are stable and repeatable through standardisation. The meticulous monitoring of results of these processes is not sufficient to become excellent. Deming stated that ‘A goal without a method for reaching it is useless... But setting goals without describing how they are going to be accomplished is a common practice among American managers.’ (Walton, 1986, p. 77). According to Imai (1997) the predominantly results-driven thinking in the West must be replaced by a process-oriented approach. Liker later mentions that the ‘right process will produce the right results’ (2006).

In other words, standardisation within and around processes is a crucial element in achieving consistently good results. This requires documented standards being followed and improved by intrinsically motivated faculty members and administrators. This should occur after they have been trained correctly in the application of these standards, based on the *Job Instruction* method (Training Within Industry Service,

1944). The important role of ensuring that standards are available, followed, and improved, lies with the leadership team.

According to Imai (1997) stability must be achieved in five key areas in a process to improve results (referred to as the **5Ms**):

- (i) **Manual power (people)** where low staff turnover is a competitive advantage because highly skilled and experienced people are staying for lengthy periods because they find fulfilment in doing value-added work. Stability with people means they know what is expected of them and they have the skills to do their work based on clear standards. Moreover, when a stable workforce finds encouragement in the respectful interaction with students, colleagues, and other stakeholders, they will care more about students, management and even national pedagogical policies. Creativity enters the workplace where the culture is conducive to learning and personal development. This will be reflected in the performance results of students and educators.
- (ii) **Machines/equipment/software/educational systems** must be reliable and well maintained so they are always immediately available to create value for students (or other customers), whether in the hands of educators, or through self-service by students. Educators must not be hoodwinked by thinking that technology and artificial intelligence will make processes better (Emiliani, 2015a). It is better processes that will enable us to design fit-for-purpose technology solutions. Huge financial expenses and massive time waste can follow the premature introduction of technology (or the next level of technological advances).
- (iii) **Materials** can include educational training resources used to add value for the students (and other stakeholders). It should be easy for students and teaching staff to access learning materials when they require it. Pursuing quality content is required. However, compared with manufacturing, the material flowing through the educational process is primarily the student. And this is where it becomes challenging for education because the students flowing through our teaching processes vary dramatically in 'quality' and consistency, unlike most factories. Factors like low income families, family violence, materialistic affluence, cultural differences, language barriers, single-parent families and the breakdown of the traditional family unit, makes the 'handling of this raw material' unique. Nonetheless, this is also where many passionate educators find their fulfilment and this must be celebrated and supported. Recent attempts to improve consistency with the quality of students entering the educational system include better screening for college-readiness in the USA (Davis Educational Foundation, 2012) and focusing on developing sufficient levels of essential reading-related skills of new school entrants in New Zealand (Tunmer et al., 2013).
- (iv) **Methods or processes** include teaching processes to develop students' skills and knowledge. It also entails administrative processes like enrolment, library access, etc. The vital processes must be identified and standardised to improve consistency in the classroom. It must be noted that standardisation does not

imply rigidity; just the opposite. Cooke stated in 1910 that a standard is simply the best method at the time the standard is created. The purpose of a standard is to make work easier, better, faster and cheaper; not to make the workplace unbearable and frustrating. To avoid entropy, stable processes must constantly be improved by staff and customer, encouraged by senior leaders.

- (v) **Measurement** of our performance and our people will help to show the gaps so improvements can be made. Without measurement, how will an organisation know if it is improving or deteriorating? Surely, it is difficult, and sometimes almost impossible, to measure certain aspects of the workplace, like feelings of loyalty or pride in one's work. Deming's management method even states that organisations should eliminate numerical quotas (Walton, 1986) when not backed by a stable process. Emiliani also warns against the use of metrics without a deep reflection on the behaviour it will create (2005). Measurements should always help a team to see the gaps in performance so that corrective action can be taken.

If any of these five areas are under strain, unreliable or inconsistent, then quality educational outcomes will be very difficult to achieve. Consistently good results (and a good reputation) demands consistent and robust processes. Senior management must set the environment where there is a continuous focus by everybody, every day, everywhere to follow and enhance standards in teaching and administrative processes. There is no point in expecting certain outcomes (whether quality, cost, or speed) if the underlying processes to achieve these targets have not been defined and standardised.

On leadership in higher education Emiliani states: "It is common to hear senior managers say 'we looked at the numbers' to justify the cuts... but almost never do they say 'we looked at the process' to understand and eliminate costs that customers do not value." (2005, p. 4). Coimbra states it boldly: 'It is this focus on improving process detail that will bring extraordinary results.' (2009, p. 8).

## 2.7 Pull and Flow

The ultimate objective of the *kaizen* methodology is to make services (e.g. information and people), and materials flow when the customer needs it; when they 'pull' the information, knowledge, service, or material from the educator or education process. It is also known as *just-in-time* processes. Delays in a process easily turn into more *muda* as previously discussed; when 'flow' is hindered, value (for internal and external customers) diminishes. Management reports should be prepared in such a manner that it flows immediately when needed by the head of a faculty two days after the semester concluded; no waiting. The process steps should also be synchronised to ensure a continuous flow from one person to the next. Examination results, for instance, should flow to students in a timely manner when it is time to pull them from the student management system. The timely feedback on assignments is also impor-

tant to foster learning. If there are delays in providing feedback to students, a learning opportunity goes begging and the quality of their education can be compromised.

To introduce pull-flow processes requires a truly student-centric system. A pull-flow system could incorporate the availability of material when students ask for it or when they are ready for it. The opposite of ‘pull’ is a ‘push’ system ‘wherein faculty design courses with the information that they think students need to know.’ (Emiliani, 2016, p. 8). More research and experiments are needed in this area although blended courses and on-line learning starts to fulfil that need.

When introducing the flow methodology we initially endeavour to make information, people, and materials flow by reducing the non-value-added activities in processes, and then we make the processes flow faster. Implementing flow is an advanced *kaizen* methodology but flow can already be improved through the initial elimination of *muda*. Every improvement should improve the flow of the process.

All previous six *kaizen* principles are supportive of achieving flow throughout processes.

### 3 *Kaizen* and Education

The efficiency expert, Frederick Taylor, raised his concern about the low quality and high cost of university qualifications early in the twentieth century. He was especially concerned about the poor work ethic of graduates and their disrespectful attitudes towards workers (Emiliani, 2015a). It sounds frighteningly familiar more than a century later—it seems that education is still facing the same issues.

Morris Cooke added his voice soon after Taylor in a report entitled *Academic and Industrial Efficiency* (Cooke, 1910). His research indicated that tertiary institutions incorrectly regarded themselves as unique and very different from other organisations, creating a mind-set of superiority and exclusivity. This has prevailed in many educational institutions with an unwillingness to learn from the practices of organisations in other sectors. This non-scientific thinking goes against the essence of pedagogy and it prohibits learning and improvement. Cooke also indicated that teachers were not spending enough time on activities adding value to students; their administrative work took their focus away from their main purpose. His work was largely ignored at the time. Even until today, applying business excellence approaches has been given little attention in education. It might just be that the same scientific method applied to academic research has not been put to practice in the administrative or teaching processes of many educational organisations.

Since then various voices have pushed for improvement in education especially since the popularisation of *kaizen* and Lean in the 1980s and 1990s. However, this chapter will not explore this pathway.

Emiliani (a seasoned Lean practitioner, turned academic) published a research paper on the application of *kaizen* in business degree programmes (2005). He concluded that if *kaizen* is applied correctly, it can rapidly improve courses and is an

organisational excellence approach that can create value for all stakeholders, something the traditional management style cannot usually emulate.

### ***3.1 Where and How to Start with Kaizen in Education?***

Senior leaders, educators and employees in this industry should consider a range of factors when transforming from a traditional managed institution to a *kaizen* culture of excellence.

#### ***3.1.1 Kaizen Leadership***

The *kaizen* journey will be doomed to failure if *kaizen* thinking and *kaizen* habits are not developed first in senior leaders. This might take time but it will prevent numerous inefficient and fake *kaizen* activities in the medium term and failure to improve in the long-run.

*Kaizen* leadership is critical in initiating, planning, leading and sustaining better ways of working. Imai (1997) refers to the two functions of organisational leadership: (i) ensure standards are maintained, and (ii) promote the enhancement of the current standards through structured problem solving by all employees. Although leaders are usually well qualified, their decisions can do serious harm to processes, people, the environment, and society. Emiliani says: ‘While we may think of leadership as intelligent, thoughtful, and capable, it would be wise to recognise it as an error-prone activity whose quality is normally very poor.’ (2015a, p. 56). Without strong, ethical leadership, any organisation will suffer.

*Kaizen* leadership is vastly different from the mainly results-driven traditional Western management style. *Kaizen* leaders coach and model the productive behaviours that will deliver sustainable and repeatable results. They do not tell, force, bully or threaten people into compliance. They are leading by example at the *gemba*, inspiring people but also applying discipline within teams through the use of the *kaizen* tools. ‘When shaping a culture, the desired core beliefs and behaviours need to be defined and spoken explicitly. This begins with humility, alignment, and a safe environment’ (Miller et al., 2014, p. 87).

The role of *kaizen* leadership is to support frontline staff; the educators working at the coalface of the education system. In a *kaizen* organisation the staff do not serve the needs of leadership; leadership serves and enables employees to ensure value is being delivered to the customer, especially the student. They encourage and coach the use of structured problem solving techniques in a respectful manner. ‘The expectation of leadership at Toyota is to effectively develop people so that performance results are constantly improving.’ (Liker & Meier, 2006, p. 221). Hence their mantra, ‘We don’t just build cars, we build people’ (ibid., p. 242). *Kaizen* leadership must be expressed through *leader standard work*; formalising disciplined leadership activities. This

standardisation of the leaders' responsibilities supports the development of the new *kaizen* habits of senior leaders, middle management and frontline leaders.

Kevin Meyer recorded his observation on *kaizen* leadership during a Lean study tour to the Toyota plant in Kyushu, Japan: "Leadership at Toyota is humble. Fujio Cho [former President of TMC] has said 'lead as if you have no power.' After seeing this facility, you truly understand that concept. Toyota is a principle, a system that just happens to have a leader." (2008). The role of leadership in education systems is to ensure that sector-specific problems are dealt with at the frontline, engaging and developing the people dealing with these daily frustrations to help resolve them. There must be a deliberate break with the self-serving leadership tradition that has been prevalent in many organisations (Emiliani, 2015a). In education there is a huge need to bury egos as this will not serve the people within the system, whether they are colleagues or learners.

Quality interpersonal relationships are deeply rooted in the *kaizen* philosophy. An excellent team is more than just following policies, processes, and procedures. Leaders create an environment where people can flourish, build confidence and expand their self-esteem. The pessimists will say it is unrealistic and unachievable. The optimists in education will ask 'Why can't it be done?'

The education sector should learn from the failures of Lean in other settings. One observation is that the leadership role in *kaizen* cannot be abdicated to a *kaizen* champion or a business excellence team. The *kaizen* leadership capabilities must first be developed to enable a committed journey.

Commitment to developing excellence through resilience must be developed in leaders to ensure continuation of *kaizen* during the change management process. An awareness of the five stages of dealing with change or loss can provide insight to leaders on how to support their school or university more effectively and efficiently. Kubler-Ross introduced the following stages: (i) denial; (ii) anger; (iii) bargaining; (iv) depression; and (v) acceptance (Connelly, 2016). Teams and individuals might get stuck in one of the stages and this can impede *kaizen* without the guidance of leaders.

Senior leaders and middle management must include *kaizen* as a strategy (Imai, 1986) and communicate the vision, mission, values and principles, continuously throughout the organisation and set up processes to give regular feedback on the progress. Setting up effective team boards, with simplified KPIs displayed can greatly enhance the quality and frequency of this communication. A mutually-agreed transformation roadmap can also help to clarify the journey. Policy deployment (*Hoshin Kanri*) should be cascaded throughout the organisation with feedback provided from all organisational levels to improve accountability, based on a deeper understanding of the purpose and direction of the organisation's journey. Senior decision-makers should therefore move away from a 'top-down' approach and include more of a 'bottom-up' process to ensure their expectations, and those from *gemba* people, are aligned. This will realistically happen progressively as the *kaizen* capabilities of senior leaders are developing.

### 3.1.2 Create a Lighthouse of Excellence

Carefully choose a team or department that is willing and committed to make improvements when embarking on this delicate *kaizen* transformation journey. They must be open for change. Senior leaders and *kaizen* champions cannot waste precious energy and other resources at the beginning of the *kaizen* journey in trying to convince the nay-sayers. In general, do not start with the toughest team. To make the workplace better can be arduous in the beginning, so, be easy on the people planning, organising, leading, and supporting *kaizen*. However, *kaizen* is often needed because of critical issues in certain departments and these will have to be addressed first whether these teams are ready for *kaizen* or not. Wisdom, respect for people and process, and transparent communication, is pivotal to progress under these circumstances.

Solving real problems to reach clear goals must always be the motivation for doing *kaizen*. Do not be tempted to embark on an all-encompassing *kaizen* training programme from the outset if you have not defined what your major problems are. First determine clearly what the actual problems are. Then prioritise your efforts and demonstrate and promote a ‘can-do’ attitude whereby problems are almost celebrated because these can germinate into improvements. Start small but get real improvements to showcase the benefits of *kaizen* rather than trying to improve everything and everyone from the onset.

One of the usual problems when starting with *kaizen* is the lack of time to do *kaizen* (Miller et al., 2014). Creating time for improvements can be one of the first problems to be solved by teachers, administrators, and senior leaders. An easy way to achieve this is to identify and remove *muda* immediately (Imai, 1997). Senior leaders can lead staff on *gemba* visits to formally identify the 3Ms (*muda*, *mura*, and *muri*) and to enable waste reduction.

When real obstacles are removed, making processes simpler, better, faster and cheaper (Phillips, 2014), people will regain hope and start to trust the *kaizen* approach. This increased intrinsic motivation of teachers, professors, coaches and administrators can become the foundation of further improvements.

### 3.1.3 Resistance to Change

When embarking on a change management excursion, resistance to change is usually high on the agenda. It is not uncommon to hear that ‘we have tried this before’, or ‘this will soon go away – it’s just another flavour of the month.’ People sometimes actively resist the envisaged changes. These are real concerns and should be dealt with transparently.

Reasons for this resistance might include the fear of loss of control, uncertainty, past resentments and disillusionments with leaders and colleagues, the loss of face, laziness, concerns about own competency, uncertainty, protecting comfort, lack of trust, and the list goes on... (Kanter, 2014). However, it is vital to uncover the root causes of these deeper seated problems as an organisation progresses with *kaizen*. This might be one of the most challenging problems to deal with but as long as it



is hidden, ignored or denied (often by senior leaders as they do want to be implicated) *kaizen* will be smothered. Only when leaders are humble enough to also be accountable for deep-seated problems, will the rapid progress be made. The servant-leadership model will greatly enhance outcomes.

The objection to applying *kaizen* in sectors outside of manufacturing is sometimes expressed. This is often indicative of a narrow-mindedness or being ill informed. There is a vast richness in understanding the principles driving organisational excellence. These can be applied in any industry, sector, cultural and religious group, sports team, and in personal life. As Toussaint, former CEO of ThedaCare, highlights his healthcare team's learning from visiting a Lean factory: 'Sick people were not snow blowers. The snow blowers were in many ways treated better. Work on each snow blower was designed to happen efficiently, without waiting between procedures, and with every employee understanding his or her role. Quality had improved dramatically. There was a lot to learn on that shop floor.' (2010, p. 14).

Being aware of one's paradigm can be very helpful in becoming more open to new ideas. A paradigm is the way a person or group sees the world based on their values, beliefs, and strengthened by their standards, habits, and past experiences (Coimbra, 2009). Academics, educators, senior leadership teams or teachers must apply their critical thinking skills and be open-minded about the application of *kaizen* in education. An unwillingness to explore and learn from others, is not only unscientific and arrogant, but also dangerous in an ever-changing environment.

Nonetheless, it must be said that the 'copy-and-paste' approach to implementing the *kaizen* tools is damaging to this proven philosophy and will result in resistance and resentment. A *kaizen* system cannot be copied; the spirit of an organisation cannot be replicated. It must be developed; continuously.

To experience doubt about a new approach, even resistance, is a normal response to a perceived threat. It is a built-in defence-mechanism that can be indicative of people caring about their work and their customers. Instead of resisting the resistors, leadership should embrace this. Educators cannot be pushed blind-folded into the unknown. Leadership must lead them with respect onto a common-found better pathway, whereby people continuously get a better understanding of what the purpose of the journey is. Policy deployment and continuous communication about the organisational goals is pivotal to minimise resistance to change.

Scholtes et al. succinctly summarises one of the laws of organisational transformation: 'People don't resist change, they resist being changed.' (2003, p. 7). *Kaizen* is never done to people. They must be led and guided to a point where they understand the purpose of the improvements. They must be included in making changes for the better that will be meaningful to them. This requires patience, endurance, and humility from leaders and managers.

Once people have been genuinely included in determining what 'better' is, and the 'why' of the transformation, persistent resistance to change has to be dealt with decisively. Everyone must know there is a strong commitment from leadership that *kaizen* is the way forward for the organisation; to become 'better' is non-negotiable. It is often better for the organisation if the persistent resistors leave sooner rather than later. These people often impede the development of others and halter process

performance. But, always deal with these people in a respectful, kind way. Servant-leadership does not imply weakness or tolerating disrespect.

## 4 Conclusion

Although there might be challenges and obstacles to implementing *kaizen* in education, it can deliver results traditional management styles cannot achieve due to its holistic and respectful approach. However, it will require strong servant-leadership, humility and a willingness to explore, experiment, and learn about the proven field of *kaizen*. Education should learn from the bountiful *kaizen* experiences (failures and successes), knowledge, and skills available, especially from other sectors. Scientific thinking should not only be applied to curriculum development and research, but, also to the processes and people employed in creating value for students and other stakeholders.

To enable the creation of a culture of excellence, everyone in an educational institution must understand and apply the seven principles, beliefs, and values every day, everywhere. Developing and cascading a clear strategy and policies to all levels is a primary responsibility of senior leaders. It should focus on making processes easier, better, faster and cheaper; in that sequence. Goals related to people development and their motivational levels should also be monitored alongside the growth aspirations of the institution.

*Kaizen* is a culture of excellence; not individual acts of brilliance or even the use of *kaizen* methods to make education better. It is the continuous improvement of a holistic system, based on the seven *kaizen* principles, beliefs, values and behaviours, made explicit through the *kaizen* capabilities of leaders at all levels. Creating *kaizen* lighthouses of excellence can overcome resistance to change when supported by respectful, caring leaders. Educational excellence will occur when ‘the concept of *kaizen* is so deeply ingrained in the minds of both managers and workers that they often do not even realise that they are thinking *kaizen*.’ (Imai, 1986, p. xxix)

## References

- Toyota Production System Manual. (2017). English translation: Warren, M. Retrieved March 30, 2018, from <https://www.linkedin.com/pulse/tps-manual-chapter-2-section-3-mark-warren/>.
- Alliance for Excellent Education. (n.d). *Teacher attrition costs United States up to \$2.2 billion annually*. Retrieved March 30, 2018, from <https://all4ed.org/press/teacher-attrition-costs-united-states-up-to-2-2-billion-annually-says-new-alliance-report/>.
- Allpress, K. (2018, 23 January). Fear teacher shortage may affect subjects. Stuff. Retrieved March 25, 2018, from <https://www.stuff.co.nz/timaru-herald/news/100709425/fear-teacher-shortage-may-affect-subjects>.

- Ballé, M. (2010, November 22). Lean = TPS (Kaizen + Respect). Gemba Coach. Lean Enterprise Institute. Retrieved March 25, 2018, from <https://www.lean.org/balle/DisplayObject.cfm?o=1698>.
- Bigham, G., & Ray, J. (2012). The influence of local politics on educational decisions. *Current Issues in Education*, 15(2). Phoenix: Arizona State University.
- Balanced Scorecard Institute. (n.d.). Balanced scorecard basics. Retrieved March 30, 2018, from <https://www.balancedscorecard.org/BSC-Basics/About-the-Balanced-Scorecard>.
- Cameron, W. B. (1963). *Informal sociology: A casual introduction to sociological thinking. Volume 1 of Studies in sociology*. Michigan: Random House.
- Coimbra, E. A. (2009). *Total flow management. Achieving excellence with kaizen and lean supply chains*. Zug, Switzerland: Kaizen Institute Consulting Group Ltd.
- Cole, K. (2001). *Supervision: The theory and practice of first-line management* (2nd ed.). Frenchs Forest, NSW, Australia: Pearson.
- Connelly, M. (2016, 23 November). Kubler-Ross five-stage model. Change Management Coach. Retrieved December 10, 2017, from <http://www.change-management-coach.com/kubler-ross.html>.
- Cooke, M. (1910). Academic and industrial efficiency; a report to the Carnegie foundation for the advancement of teaching. *Academic and Industrial Efficiency Bulletin, Number Five*. The Carnegie Foundation for the Advancement of Teaching. The Ontario Institute for Studies in Education. Retrieved December 10, 2017, from [http://www.archive.org/stream/academicindustr05cookuoft/academicindustr05cookuoft\\_djvu.txt](http://www.archive.org/stream/academicindustr05cookuoft/academicindustr05cookuoft_djvu.txt).
- Davis Educational Foundation. (2012). *An inquiry into the rising cost of higher education. Summary of responses from seventy college and university presidents*. Retrieved from <http://www.davisfoundations.org/def>. Yarmouth, USA.
- Deming, W. E. (1986). *Out of the crisis*. Cambridge, MA: MIT Center for Advanced Engineering Study.
- Emiliani, M. L. (2005). Using kaizen to improve graduate business school degree programs. *Quality Assurance in Education*, 13(1), 37–52. <https://doi.org/10.1108/09684880510578641>.
- Emiliani, M. L. (2015a). *Lean university. A guide to renewal and prosperity*. Connecticut: The CLBM.
- Emiliani, M. L. (2015b). *Lean teaching. A guide to becoming a better teacher*. Connecticut: The CLBM.
- Emiliani, M. L. (2015c). *Lean is not mean. 69 practical lessons in lean leadership*. Wethersfield, Connecticut: The CLBM, LLC.
- Emiliani, M. L. (2016). *Evolution in lean teaching*. New Britain: Central Connecticut State University.
- Emiliani, M. L. (2017a, January 2). *Lean: Past, present, and future*. Bob Emiliani blog. Retrieved May 10, 2017, from <http://www.bobemiliani.com/lean-past-present-and-future/>.
- Emiliani, M. L. (2017b, October). *A study of executive resistance to lean* (working paper). New Britain: School of Engineering, Science, and Technology, Central Connecticut State University.
- Graban, M. (2009). *Lean hospitals. improving quality, patient safety, and employee satisfaction*. New York: Productivity Press.
- Graban, M. (2007, March 21). *Lean of "L.A.M.E."?* Mark Graban's Lean Blog. Retrieved November 25, 2017, from <https://www.leanblog.org/2007/03/lean-or-lame/>.
- Green, V. A., Harcourt, S., Mattioni, L., & Prior, T. (2013). *Bullying in New Zealand Schools: A final report*. Wellington, New Zealand: University of Victoria.
- Hellriegel, D., Jackson, S. E., & Slocum, J. W. (2002). *Management: A competency-based approach* (9th ed.). Cincinnati, Ohio: South-Western/Thomson Learning.
- Imai, M. (1986). *Kaizen. The key to Japan's competitive success*. United States of America: McGraw-Hill.
- Imai, M. (1997). *Gemba kaizen. A commonsense, low-cost approach to management*. Singapore: McGraw-Hill Book Co.

- Lambert, K. (n.d.). Why our teachers are leaving. Education World. Retrieved March 30, 2018, from <http://www.educationworld.com/why-our-teachers-are-leaving>.
- Mind Warriors Limited. (2009). *Jolt challenge. The self intelligence experience*. Auckland: Star Books.
- Kaizen Institute New Zealand. (n.d.). Retrieved November 19, 2017, from <https://nz.kaizen.com/se-kaizen.html>.
- Kaizen Institute USA. (2018, 20 March). Cash flow vs. cost reduction. Retrieved March 29, 2018, from <https://us.kaizen.com/blog/post/2018/03/20/cash-flow-vs-cost-reduction.html>.
- Kanter, R. M. (2014, 25 November) Ten reasons people resist change. *Harvard Business Review*. Retrieved November 9, 2017, from <https://hbr.org/2012/09/ten-reasons-people-resist-change>.
- Krafcik, J. F. (1988). Triumph of the lean production system. Massachusetts Institute of Technology. *Sloan Management Review: Fall, 30*(1), 41–52.
- Liker, J. K. & Meier, D. (2006). *The Toyota Way Fieldbook. A practical guide for implementing Toyota's 4Ps*. USA: The McGraw-Hill Companies, Inc..
- Meyer, K. (2008, 28 October). JKE Day 1: Toyota Kyushu—The Manufacturing Ballet. Kevin Meyer Blog. Retrieved June 16, 2017 from <http://kevinmeyer.com/blog/2008/10/jke-day-1-toyota-kyushu.html>.
- Millar, S., & Theunissen, C. A. (2008). *Managing organisations in New Zealand* (3rd ed.). North Shore, Auckland: Pearson Education NZ.
- Miller, J., Wroblewski, M., & Villafuerte, J. (2014). *Creating a kaizen culture: Align the organization, achieve breakthrough results, and sustain the gains*. USA: McGraw-Hill Education.
- Mirsky, J. (1998, 20 July). Eli Whitney. American Inventor and Manufacturer. Encyclopaedia Britannica. Retrieved October 22, 2017, from <https://www.britannica.com/biography/Eli-Whitney>.
- Mullenlowe U.S. (2010, 1 April). Retrieved November 5, 2017, from <https://us.mullenlowe.com/zappos-happy-people-making-people-happy/>.
- Net Promoter Score (n.d.). Retrieved December 14, 2017, from <https://www.netpromoter.com/known/>.
- Organisation for Economic Co-operation and Development. (2010). *Overcoming school failure: Policies that work. OECD Project Description*. Retrieved March 30, 2018 from <http://www.oecd.org/education/school/45171670.pdf>.
- Phillips, P. (2014, 2 July) *What makes your product valuable? Journey on the value stream—Part 3 on Organizational improvement in lean technology transformation*. Retrieved December 10, 2017, from <http://leantechnologytransformation.blogspot.com.nz/2013/02/what-makes-your-product-valuable.html>.
- Rother, M., & Shook, J. (1998). *Learning to see*. Cambridge, MA, USA: Lean Enterprise Institute.
- Shingo, S. (1959) (2007—English Translation). *Kaizen and the art of creative thinking*. Tokyo: Hakuto-Shobo Publishing Company.
- Shingo, S. (1988). *Non-stock production: The Shingo system of continuous improvement* (1st ed.). U.S.A: Productivity Press.
- Scholtes, P., Joiner, B. L., & Streibel, B. J. (2003). *The team handbook* (3rd ed.). USA: Oriel Inc.
- The Princeton Review. (n.d.). *Homework wars: High school workloads, student stress, and how parents can help*. Retrieved March 30, 2018, from <https://www.princetonreview.com/college-advice/homework-wars>.
- Toussaint, J. & Gerard, R. A. (2010). *On the mend: Revolutionizing healthcare to save lives and transform the industry*. Cambridge, MA, U.S.A: Lean Enterprise Institute.
- Toyota Global Website. (n.d.). Retrieved December 3, 2017, from [http://www.toyota-global.com/company/history\\_of\\_toyota/](http://www.toyota-global.com/company/history_of_toyota/).
- Training Within Industry Service. (1944). *Job instruction: Session outline and reference material*. Washington, D.C.: Bureau of Training, War Manpower Commission.
- Tunmer, W. E., Chapman, J. W., Greaney, K. T., Prochnow, J. E., & Arrow, A. W. (2013). *Why the New Zealand National Literacy strategy has failed and what can be done about it: Evidence from the progress in International Reading Literacy Study (PIRLS) 2011 and reading recovery monitoring reports*. Auckland: Massey University of Education.

- Walton, Mary. (1986). *The Deming management method*. New York: The Berkley Publishing Group.
- Weimer, M. (2018). Confronting the myth of multitasking: A collection of tools and resources. Faculty Focus. Retrieved Accessed March 29, 2018 from <https://www.facultyfocus.com/resources/teaching-strategies-techniques/motivating-students/confronting-myths-multitasking-collecti-on-tools-resources/>.
- Willis, J. (2012, October 16). Deming to DevOps (Part 1). IT Revolution. Retrieved May 12, 2017, from <http://itrevolution.com/deming-to-devops-part-1/>.
- Womack, J. P., Jones, D. T., & Roos, D. (1990). *The machine that changed the world*. New York: Free Press.
- Womack, J. P., & Jones, D. T. (1996). *Lean thinking: Banish waste and create wealth in your corporation*. New York: Simon & Schuster.
- Womack, J. P. (2016, August 29). Womack—Sustaining gains must become a focus of management. Planet Lean. Retrieved April 18, 2017, from <http://planet-lean.com/womack-sustaining-gains-must-become-a-focus-of-management>.
- Wroblewski, M. (2006, 28 September). Lean manufacturing epiphany. Got Boondoggle? Retrieved June 18, 2017, from <http://gotboondoggle.blogspot.co.nz/2006/09/lean-manufacturing-epiphany.html>.

**Part II**  
**Agile Methods in the School Classroom**

# Transforming Education with eduScrum



Willy Wijnands and Alisa Stolze

**Abstract** In this practitioner chapter, the author (the originator of eduScrum) explains the motivation for creating eduScrum, outlines its core practices, processes and artefacts, and includes an external perspective on how eduScrum works based on a series of conversations with students, as well as two experience reports from students who have experienced eduScrum in practice. From the motivation that education needs to change to meet the needs of twenty-first-century society, this chapter addresses the various ways that adapting Scrum processes to the classroom enables students to become more independent and self-directed in their learning. Student experience reported in the chapter suggests that the positive aspects of eduScrum reinforce each other over time, so that students become increasingly comfortable with this style of learning as they progress through year levels.

**Keywords** Agile · Education · eduScrum · Personal development · Trust

## 1 Introduction: Change, Education, and Agile Methods

The world is changing very fast and we must adapt to this change. Unfortunately, the current educational system is obsolete. We are still teaching as we did in the age of industrialization, which used to make perfect sense but is no longer applicable (Martin, 1995). This creates a gap between the educational offer and the market requirements, as nowadays we need to not only educate workers, but leaders and highly adaptive thinkers. There are three fundamentals underlying this change:

---

W. Wijnands (✉)  
eduScrum, Alphen aan den Rijn, The Netherlands  
e-mail: [willywijnands@gmail.com](mailto:willywijnands@gmail.com)

A. Stolze  
eduScrum, Berlin, Germany  
e-mail: [alisa.stolze@scrum-events.de](mailto:alisa.stolze@scrum-events.de)

© Springer Nature Singapore Pte Ltd. 2019  
D. Parsons and K. MacCallum (eds.), *Agile and Lean Concepts for Teaching and Learning*, [https://doi.org/10.1007/978-981-13-2751-3\\_5](https://doi.org/10.1007/978-981-13-2751-3_5)

1. Less need to examine knowledge retention, instead more focus on developing the students' twenty-first-century skills such as collaboration, communication, critical thinking, creativity, ICT skills, etc.
2. Changing demands of society. Knowing your weaknesses as well as your qualities becomes even more important as the time of the individual at the workplace is over, with increasing emphasis on teamwork and collaboration.
3. Going beyond the teaching of the subject to help students to develop as human beings and let them make progress in their personal qualities. The teacher is no longer the source of knowledge.

The attitudes of students also need to change. We have to help students change the attitudes from the way they have been taught before. Many students lie back as if they were in a movie theatre. Students must leave this cinema attitude behind.

So, could eduScrum, as outlined in this chapter, become the connecting element? We believe so. In eduScrum, the students are working together in teams in an active, effective and efficient way and have more fun. This is achieved by giving the students ownership of their own learning process, but most important, trusting them. The students take responsibility for what they do because they are given the freedom and the space they need. The effect is that students are more engaged, more productive and their results are better. They discover who they are and what their abilities are.

## ***1.1 The Origins of eduScrum***

In the book 'The Art of Doing Twice the Work in Half the Time', Jeff Sutherland, the inventor of Scrum (Sutherland & Sutherland, 2013), suggests that Scrum, a well-known agile method for software development, can be multi-faceted, assisting in many areas of life, not just the business world. Sutherland also gives an account of Scrum used in education, specifically citing a chemistry class in the Netherlands, where eduScrum began.

The author was teaching science at Ashram College in Alphen aan den Rijn, and was introduced to Scrum by Mark Reijn, a software engineer at the company Schuberg Philis in the Netherlands. Excited by the Scrum principles, he began implementing a modified version of Scrum in 2011 with his students ranging from age twelve to age eighteen, with impressive results. His students became excited to work as a team on their projects as they developed life skills such as taking ownership of their work and learning to be focused, hardworking and motivated. In order to assist other teachers in implementing that special version of Scrum modified specifically for educators in their classrooms, the author founded eduScrum and produced the eduScrum guide (Delhij, van Solingen, & Wijnands, 2015).



## 1.2 What Is eduScrum?

eduScrum is an edit of Scrum (Schwaber & Sutherland, 2017) a framework for an active, collaborative, co-creative education process. It allows students to make assignments according to a fixed rhythm. They plan their own activities and keep track of their own progress. The teacher determines the assignments, coaches and gives advice.

In eduScrum we go from teacher-driven education to student-driven and student-organized education. The teacher determines the why and the what, the students determine the how. With eduScrum, the students own their own learning process. This results in intrinsic motivation, fun, personal growth and better results. Learning is the key element: effective and efficient learning, learning to cooperate better, learning to get to know oneself better, learning to be co-creative. Students work together in an energetic, targeted way. This way of working generates pleasure, power and accountability, the work goes faster and the results are better.

eduScrum students are stimulated to develop into valuable members of a team and develop a mindset that aims for constant improvement. They pass through a positive personal development. eduScrum is a ground-breaking way of education, where personalized learning has a very important role, like the 4 C's of learning and innovation skills: Creativity and Innovation, Critical Thinking and Problem Solving, Communication and Collaboration (P21, 2016). eduScrum gives students a free rein to take agency over their learning.

One of the modifications made to Scrum was to add a 'Definition of Fun' to the process, as well as a Definition of Doing instead of Definition of Done. This is because fun is an important motivator for students, and is an essential part of improving learning. This coincides with the happiness factor that Sutherland and Sutherland (2013) explain:

Our day-to-day life is mostly made up of journeys. We don't summit peaks every day...Most of our days are taken up with striving toward our goals...if we get rewarded only for the results, not the process, we're going to be pretty miserable.

In the learning process it is the same for students. eduScrum captures the importance of enjoying and placing value in the journey. It provides the framework for working effectively as a team, and increases productivity exponentially. In education, teachers are using eduScrum to teach accountability, self-motivation, team collaboration and proper time management skills to their students, using the work on the actual school curriculum as a vehicle.

The secret of eduScrum is 'Ownership'. Students are given ownership of their own learning process, but most important trust. As Covey (2006) states, the ability to create, preserve and restore trust has become one of the most important skills of today. The students take accountability for what they do because they are given the freedom and the space they need. The effect is that students are more engaged, more productive and their results are better. Freedom is the acknowledgement of borders. The teacher is no longer responsible for the learning process of their students, but delegates

that responsibility to the students—their role changes into a coach, facilitator and counsellor.

The collaborative nature of eduScrum leads to personal development, based on four building blocks: trust, communication, involvement and accountability. Cooperation requires mutual trust between students. In addition, communication is very important so that they learn to be themselves and dare to say what they think. These two building blocks provide involvement within the team, which in turn leads to accountability. The four building blocks refer not only to the team as a whole, but also to individual students.

eduScrum has been developed to stimulate the personal development of students. This is an active collaborative form in which students in teams make assignments according to a fixed rhythm. They themselves determine their activities and keep track of their own progress. The teacher determines the assignments and provides support to a class, team or individual student, where necessary. Through this method, students themselves think about how they want to learn.

## **2 An Outline of eduScrum**

This section outlines the key components and practices of eduScrum, involving teams, process, planning and reflection.

### ***2.1 Teams***

In eduScrum, work is done in teams of four to five students. eduScrum teams are self-organized, which means that they themselves determine how they want to work together within the set boundaries of the assignment and framework. One of the first meetings in an eduScrum project is team formation. The team formation meeting takes place at the start of a sprint. First, a team captain is appointed by the teacher, or chosen by the class. The Team Captain is not the boss, but someone who makes the team work well. Regardless of whether the team captains are chosen or designated, they choose the members of their team so that they can achieve an optimal team composition. They choose their teammates based on the qualities needed for a good project result. Lists of potential team members are anonymized, and only the gender of the relevant student is known. We do not want teams with friends, or only friends, nor teams that are single sex. Female team captains first choose a male team member, and vice versa. The order in which team captains may choose is varied as much as possible. Team captains choose teams of four or five team members. The students then choose a catchy name for their team, and make their own arrangements for how the team will work. Teams usually last for two periods. This gives students the opportunity to take action after the reflection and retrospective to improve their work. After the team formation, the planning meeting will start immediately.

eduScrum is about cooperation based on mutual trust. The team is jointly responsible for a good working atmosphere, with the team captain as the ‘oilman’ who helps and coaches the team. Team captains will need to have confidence in their teammates as well as in themselves. Students will see that if they trust each other and work together with pleasure, a good result is almost self-evident.

## ***2.2 The Teacher as Product Owner and Servant Leader to the Student Teams***

The teacher determines what student teams have to learn within which period of time. They design the projects for the student teams, set the learning goals and tell the students how they want to measure how their work has been successful, but are also there for the student teams as a servant leader to assist the teams and answer questions when needed.

When teachers want to work with eduScrum in their classrooms, they must first have trust in their students. Second, they need an open agile mindset, giving students context-content-based assignments or, even better, projects where students themselves can determine what they want to learn and, most importantly, how they want to learn. Then, the ‘why’ of learning will become normal, which fosters a life-long learning process.

## ***2.3 Start with the Why***

The teacher decides the ‘why’ and the ‘what’ of the assignment, which is crucial for a successful eduScrum project. For every teacher who wants to get the full potential from his students, the Golden Circle can be the starting point. The golden circle is a thinking model developed by Sinek (2009). Like eduScrum, it’s an important tool you can use. The message is that people do not buy what you do, but why you do it. In the same way, students are unconsciously and intrinsically not interested in what you do and how you do it, they are interested in why you do it. So, start with ‘why’, even if you do not do eduScrum. Start with the students’ ‘why’ to ask questions. Why are they in your class and ‘must’ follow your subject. Explain to them its usefulness, and how they can use and apply it. Then they know and understand why they also need to do things they do not like to do. The ‘why’ is about passion, motivation, your heart-feeling, your inner self. This is not about what people believe in, it’s what they feel.

## ***2.4 Framework and Process***

The eduScrum framework is built to support teams and teachers in collaborative learning. The basic eduScrum concept consists of one result, roles, meetings, artefacts and some simple rules. Together they form a balanced structure. Each part within the framework serves a specific purpose, and is essential for the use and success of eduScrum. In the eduScrum framework all the educational approaches come together in an organic way. Students are working in projects, where they gain knowledge and skills by working for an extended period of time to investigate and respond to an authentic, engaging and complex question or problem, actively exploring real-world problems and challenges and acquiring a deeper knowledge. This gives their work a greater purpose. Students must acquire content knowledge themselves, not through the less engaging method of direct instruction. Thus, students guide their own learning, add their own meaning and experiences, dig into the material, and actively engage with the content.

## ***2.5 eduScrum Sprint***

At the heart of eduScrum is the Sprint. Doing Sprints in eduScrum means student teams work together in short cycles (sprints) according to the Plan Do Check Act (PDCA) learning and improvement cycle (Moen & Norman, 2009).

The sprint is the teacher-defined time frame within which a certain amount of work has to be done for a context-concept-based project, in which one or more learning objectives will be realized. In school education, sprints last no longer than approximately 2 months. Every sprint begins with the planning of the work to be done for the project, and ends with a review of the learning and working results, as well as a retrospective on team performance and personal development.

In the sprint, teams work together to achieve the goal set. Usually a sprint is divided into sprint reviews after approximately 3–4 h work. Students have the space, within the framework and goals set by the teacher, to find their own way to deal with their work and adapt according to their insights. However, this does not mean that teams have total freedom and are not supported in their work. On the contrary, the sprint gets structure through the rules, meetings and artefacts of the eduScrum framework. This creates a process that is highly structured: planning and executing the work, keeping track of progress towards the goals set, and looking back at team performance and personal development.

## 2.6 *Planning Meeting*

The planning meeting takes place in the first lessons of an assignment project. These are very important lessons for the students and the teams. Here they get an overview of what to expect in the coming period. It is the kick off of the project so everything should be 'Ready' at this point. The definition of 'Ready' is all the actions and steps that the teacher and the teams must have done before they can really start with the assignment/project. From the teacher's point of view, the project has to be ready and actionable for the students to plan before the planning meeting. This takes a lot of time, but this time will be paid back during the project. Dare to give time to your teams.

During planning the team altogether plans the tasks to be dealt with during the next sprint, they make their own plan transparent on an eduScrum board 'flap' and think about how to do the assignment (the flap is an overview of the process and is described in detail in the 'artefacts' section). In addition, they ask the following questions:

- What exactly needs to be done?
- How much work will this be for us?
- How do we divide the work?
- What tools do we need?

## 2.7 *Stories*

Stories are a way to describe the products that this Sprint should deliver. A story describes 'what' and 'why' it has to be done. The 'how' is not part of the story because it is left to the team in the planning meeting to decide how they want to self-organize.

During the planning meeting, the teams create these stories and think about why they are doing what they are, and what they want to learn. Stories are the things that need to be done in a Sprint, such as making assignments, experimenting, writing a report, or preparing a presentation which needs to be delivered. Stories provide a rough overview of the tasks to be dealt with.

The teams will divide the assignment project into several stories or sub-items. After looking at all of the celebration criteria for each story, each story leads to a number of tasks that need to be done. Stories are broken down into actionable, small tasks or 'To Do's. The tasks defined during the planning meeting of the team are written on post-it's. A 'To Do' Post-it always has a verb in it. It is helpful not to use words that leave too much room for interpretation, but to directly specify the 'To Do' so that it gives clarity about the Task later on in the project. Writing stories and tasks ('To Do's) promotes the students' involvement and accountability in their own learning and work process.

## **2.8 *Celebration Criteria***

The celebration criteria are defined by the teacher and explained at the beginning of each sprint. This is done to ensure the quality of what is learned. The team is responsible for adhering to the celebration criteria and defines its own tasks and measures to ensure that the celebration criteria are met. These are learning objectives, assessment lists, answer models, exam requirements, etc., and the teacher's requirements for a story. The teams put the celebration criteria on their 'flap' next to their stories.

## **2.9 *Working Agreements/Definition of Doing and Fun***

Students also make working agreements and write them in their Definition of Doing and their Definition of Fun. These are agreements about the qualities of their products and working together with pleasure to meet the celebration criteria. Each team gathers their own working agreements and puts them on their flap.

In Dutch schools a 5.5 is a positive grade on a scale from zero to ten, but for us this is unacceptable. With eduScrum the goal is to receive a grade above 6.7 to get qualitatively good students. Above 6.7 we acknowledge that the student fully understands what they have learned. Students make their own goals to achieve a grade of 6.7 or higher.

## **2.10 *Stand-up***

The stand-up is an event which takes place regularly at the beginning of each learning unit. Students run into the classroom, put their 'flap' on the wall and gather around it to start the stand-up meeting by themselves. Restricting the 'mini-meeting' to no more than 5 min duration ensures that the speed and energy of the team are high. It is important that every team member attends this meeting.

During the stand-up the following three questions are addressed by each team member to help the team synchronize:

- What have I done since the last Stand-Up?
- What will I do for my team during this lecture unit?
- Do I have any obstacles or impediments? Do I see any impediments for my team?

Particular attention should be paid to possible obstacles and problems. However, there is no deeper discussion about solving problems during a stand-up. If one team member knows the solution to a problem another team member has, he or she can briefly state that, to solve the problem directly after the stand-up so that the other teammates can start working immediately.

The team captain ensures that a stand-up is held regularly and can moderate it. However, the team itself is responsible for the performance. After a stand-up, the flap and run-up chart are immediately updated.

## ***2.11 Review***

The review is a kind of feedback meeting. In the review the teams show what they have done at the end of 3–4 working hours (classes). The team shows what it has learned in this short cycle (sprint) and receives direct feedback from the teacher. The type of the meeting is determined by the teacher and may vary from sprint to sprint. These reviews should help the team members to check the self-developed content. It is important to see if the tasks have been successfully done. Is each team member satisfied with the result? Are they still meeting the celebration criteria? If not, what does the team need to do to address that? What kind of support does the team need from the teacher?

A bigger review meeting is held at the end of a whole project. In this meeting, a team result should be delivered as well as a personal result. This could be done with a test as well as with presentations, videos, posters or anything else the teacher announces at the beginning of each project, or lets the students decide for themselves.

## ***2.12 Personal and Team Retrospective and Reflection***

At the end of a project, each team will do a retrospective on their achievement, and discuss how to do things better in the next project. Students look at their personal contribution to the team and how the team functioned as a whole. With this observation and reflection they lay the foundation for the next project.

Students also reflect on their role within the team. Each student evaluates himself or herself and his or her teammates according to their qualities and skills. This trains them in self-reflection and giving feedback to their classmates to continuously improve their own learning process and team working process.

The retrospective should never be neglected. Any postponement of the retrospective is a missed opportunity for the team and its members to improve in the next sprint and to learn more effectively and efficiently. The teacher can give a clarification about the objectives of the retrospective.

Teammates can answer the following questions to give and get feedback:

- What went well?
- How can I improve myself? How can I help others improve?
- How can we become better as a team?
- What should we no longer do?
- What actions do we take in the next sprint to improve?

- What are the facts about the quality of our work/our productivity?
- What did I teach to my team members? What did I learn from my team members?
- Why did something go well or go wrong?
- Was the time meaningfully invested?
- What should be maintained next time?

The retrospective is also a good opportunity to adapt and improve the Definition of Doing and the Definition of Fun.

A good retrospective is characterized by the fact that there is a coach who guides the team through the process. In addition, it is very important to actually develop measures that can be implemented directly in the next sprint.

### ***2.13 Team and Personal Development***

As a teacher, you have an important role to play in monitoring the quality of work reflection. The quality of this reflection process is an important factor in the personal development of the team member and individual student. This personal reflection retrospective about personality enables teams to work together more and more. These are extremely important steps in a process of becoming better as a person and as a team. This will not only increase your eduScrum process, but grow yourself as a team player and as a person.

## **3 The Flap**

The flap is a large sheet (A0 or flipchart format) of paper. Each team creates its own flap. The flap gives an overview over the process. It makes it clear to everyone what the teams have to do during the sprint. It shows what has been agreed on, and whether the work is on schedule. The flap is a chronological representation of the work during a sprint. The top line of the flap determines the project, which team the flap belongs to, and which members the team consists of. The tasks status varies from ‘still to do’ (To Do), ‘is being worked on’ (Busy) to ‘finished’ (Done). The flap must be updated regularly, so it always shows the current process of the team. Updates takes place during the stand-ups.

### ***3.1 To Do, Busy and Done***

The tasks defined during the planning meeting move according to their status from ‘still to do’ (To Do), ‘working’ (Busy) to ‘finished’ (Done).



**To Do:** The ‘To Do’ column summarizes all tasks for a sprint, what needs to be done together in the coming period. The small tasks make clear what needs to be done so that results can quickly be achieved.

**Busy:** Each Team member chooses a task from the To Do column at their own request, but always in consultation with the team. He or she moves it to the Busy column and starts working. Of course, work can be done together with other team members to achieve higher team performance. Teams need to be aware also of which items do they ALL have to know and work on.

**Done:** A task will only be completed if it meets three conditions.

- All team members must agree that the task is done.
- The result must meet the celebration criteria.
- The teacher can ask any team member questions about the outcome of the task and the team member should then be able to answer correctly.

To meet these conditions, each team member must keep the others informed about their results and help each other to really understand what has been done in the task.

In addition to the process fields (To Do, Busy, Done), the flap comprises the following areas (some of which we have already described above):

- Stories
- Celebration Criteria
- Definition of Doing
- Definition of Fun
- Run-Up Chart
- Impediments.

### ***3.2 Run-Up Chart***

The Run-Up Chart visualizes the learning progress of each team. It makes transparent to the team where it stands and whether it has to work faster or not. Each task has a number of points. Certain numbers are assigned to these tasks using Planning Poker (Grenning, 2002). Planning Poker is a game approach to assign a relative score to the individual tasks. It is a measure to make a relative estimate of the tasks to be done, as objectively as possible, according to the Fibonacci sequence. The estimation is relative because this gives quick insight into the relationships between the pieces of work in the total work backlog, to compare the effort that students will have to put into a certain number of tasks.

The total number of points that need to be processed through the sprint is added up. Now the ideal line can be drawn to the end of the sprint when all the work is done. The team velocity is the amount of work that the team can complete in one class. During each class, the amount of work is monitored. Based on this velocity, the team can estimate how much work it can handle per class.

If a card has been set to 'Done', its score can be added to the Run-Up Chart. During the stand-up, tasks might be declared Done and the Run-Up Chart can be updated. It becomes transparent to the team and the teacher whether the progress of the work is still in line.

### ***3.3 Impediments***

Impediments mean obstacles. All the obstacles that the team is struggling with are put into the 'impediments' field of the flap, where the team records everything that is preventing them from working efficiently. Faults and obstacles of all kinds are listed here. The main impediments are listed at the top. It is up to the team to remove impediments as quickly as possible, but the teacher can also possibly intervene.

## **4 The Pillars of eduScrum**

Like Scrum, eduScrum is based on the three pillars: Transparency, Investigation and Adaptation.

### ***4.1 Transparency***

The eduScrum framework is aimed at the transparency of information supporting the learning process. Transparency is needed to help the teams make the right decisions and thereby maximize the value of the result. Transparency is created in the teams and with the teacher. It also arises because it is clear who is responsible for which part of the result, by using a flap where everyone can see what is going to happen, what has been done, and whether the team still has the desired velocity. In addition, it is clear to everyone what the requirements are for executing assignments.

Students are not afraid of transparency, they are just not used to it. When students do not work within the team, the other team members do not accept it and tell the student who does not want to work in their own peer way of telling. That works very well. When a team member does not want to work in a team at all, then the teacher can take them out of that team and let them work on their own, alone. The teacher does not control the students or teams. They do this by themselves.

This is important and belongs to the values and principles of eduScrum: let the students know and feel that the transparency of abilities and working status is not used against them, but for their best learning instead. We need transparency to enable the team and the teacher to support the students in the best possible way. For this, again, you need to build trust within the teams and also, very importantly, between the teacher and the students.

## 4.2 *Investigation*

eduScrum teams must regularly review their progress towards the learning objectives. These learning objectives are not only subject matter, but also related to the functioning of the team and their personal growth. Investigations are clearly and strictly defined. These meetings take place at different times and levels in the workflow or process.

## 4.3 *Adaptation*

An adjustment occurs when a team (or teacher) determines that one or more aspects may fall outside the acceptable limits or that the result will be unacceptable. Then the planning or the work in progress will be adjusted to minimize further deviations. However, adaptation also takes place in the team's own functioning.

## 5 **eduScrum Principles and Values**

Like XP, and the manifesto for agile software development, eduScrum defines both principles and values. The principles of eduScrum are Transparency, Investigation, Adaptation, (as described above), Collaboration, Reflection, Visibility and Iteration. The eduScrum Values supporting these principles are Trust, Freedom, Commitment, Autonomy, Personal Growth, Focus, Ownership, Authenticity, Critical thinking and Creativity. eduScrum combines the Principles and Values into 'Value Principles'. Principles are fixed, while Values are in development and evaluate. There can be no real eduScrum without following these values and principles.

*Values* are social norms—they are personal, emotional, subjective and arguable. All of us have values. Principles are natural laws, they're impersonal, factual, objective and self-evident. Consequences are governed by principles and behaviour values; therefore, value principles!

So, in conclusion, principles are truths based on natural laws and the process, whereas values are about behaviour. Values may or may not have a positive impact on our lives, and on our personal development, depending on whether or not they are based on true principles.

Trust is one of the core values of eduScrum comprising self-confidence, trust in teammates and trust received from the teacher. Patrick Lencioni, the author of the book 'The Five Dysfunctions of a Team', explains why teams do not work well (Lencioni, 2002). He describes a hierarchical structure: without trust there is no good communication, discussion and confrontation, without communication no commitment, without commitment no accountability and without accountability no result. A lack of results thus often comes from lack of trust.

If we build a team, we want to avoid these pitfalls, and we must start with trust. Thus, the ‘pyramid of Lencioni’ arises, with the following five hierarchical phases.

### ***5.1 Trust***

This is the first phase of team development. A team that meets for the first time mainly consists of individuals that are not yet aligned.

### ***5.2 Communication***

If confidence has arisen and team members experience safety, teams can go to the next stage. It is communicated in a respectful way. In addition, each dares to give their opinion, and discussions are being held and opinions discussed. A team of trust can also create conflicts. It can be stormy, but it is done in a respectful way and with a common goal. If the mutual communication goes smoothly and every team member participates well, the team members get involved in the activities that the team develops.

### ***5.3 Commitment***

Because all team members talk, ideas are exchanged and each team member can give their opinion and become involved. The group of individuals will behave as a team. Synergy, cohesion and cooperation develop and they speak in their own language. The team members feel increasingly involved with the common assignment(s). This involvement creates an organic process in which they will take accountability.

### ***5.4 Accountability***

Through trust, communication and commitment team members experience a sense of accountability.

### ***5.5 Result***

The team is accountable for the success (or otherwise) of the joint end result.

## 6 Challenges of eduScrum

As with any method there can be problems with implementing eduScrum, but more often than not these problems arise from the neglect of an element of the framework. The use of the complete eduScrum framework ensures that students get the best out of themselves and their team and helps teachers to create a safe atmosphere. For young people today, self-confidence, team competencies and an agile mindset are more important than ever. eduScrum encourages them to develop into complete people who can be significant to themselves and for their team. Thus eduScrum allows them to provide a positive contribution to a better world together with others.

In the beginning, this new way of teaching feels strange to students, because they are not used to working this way. They get a lot of freedom to work together in teams. The teacher does not give homework because they do the homework by themselves. They are only given the assignment, explaining the what and the why of that assignment. Some students do not like this transparent way of working, where the peer students tell each other what to do. However, students listen better to their peers than to the teacher.

They do also have the possibility of returning to their own familiar modes of learning. Alone or in pairs. The teacher does not talk for the whole lesson, but coaches and facilitates the students. So when they want to learn in their own way, it is their choice. And this happens too, but most students want to work in teams. That is a normal, organic way of working. To be a member of a team feels good to people.

Building a good relationship with the students is the first step, based on trust and respect, it will be easy to implement eduScrum in the classroom. eduScrum is simple and that is at the same time the difficulty. Really understanding the principles and values is very important. eduScrum can give students wings! But if you don't explain the *why*, they can't fly. The other very important issue is that the teacher must have an agile mindset. Without this, they will fail.

## 7 Practitioner Report from Alisa Stolze

When I was visiting an eduScrum training in Alphen aan den Rijn in November 2016, the participating Dutch teachers and I had the opportunity to talk to eduScrum students between 14 and 17 years to ask them about their experiences with eduScrum. We were sitting together on the cozy carpet of a hotel hallway with three teams of eduScrum classes and talked. Listening to the students, I thought it was very interesting that their happiness about working with eduScrum gets significantly greater with every year of eduScrum practice. This was evident from critical voices from those new to eduScrum compared with those who had been using it for a longer period.

In those classes that had recently begun working with eduScrum we could hear several critical voices. The work with eduScrum is not actually harder than regular lessons, but it is different, because students have to plan their work by themselves.

In particular, eduScrum starters dislike having to get used to a new way of working. The planning is too much effort, they say, and it takes them too long until they can begin with the real work of learning.

'I prefer getting an assignment from a teacher, listening to the explanation and afterwards just doing it and trying to understand. All those stickers are just too much for me.' one girl told us.

Another student added: 'I even think that the stickers and the tasking is extra work. I don't need all those stickers to know what I have to do next.'

However, one grade higher, the situation is already different. Students like this way of teaching. One girl tells us, that eduScrum did not work very well for her during her first year trying. Why? Because the teacher that had been working with the class using eduScrum did not tell the pupils about why they do each step, why there is a Run-up to watch progress, for example, and why they would have a Definition of Doing and a Definition of Fun. 'So what is the Definition of Doing exactly?' one of my fellow participants asks the team. 'That something is really done. That there is nothing left to do and that we are happy with the results.' It can be so simple!

I was especially impressed by how the students explain the effort points that they create using Planning Poker in order to determine how big one item of work is:

Points don't just tell you how much of work an item is, but also how hard it is.

Planning? No problem. At the beginning it is a lot, ok, but then you estimate the points and split the effort points of work to fit the number of working lessons.

The overview of work to do or still remaining also helps the students to help each other.

Teams are based on complementing qualities: 'If I really don't get a thing I just ask my teammates and they can explain it to me.'

The eduScrum 'professionals': happy about their possibilities and freedom.

The team of students that practice eduScrum for three years now even ask following teachers to be allowed to use eduScrum for their assignments and are happy about their independence:

I don't like if people tell you what you should do. Here you are self-responsible, you can plan, you can make decisions to do more in one lesson and take it easier in another. That is great! You can always see by yourself where you are and how much work is left.

With one look at the Run Up Chart I know where I stand. It helps me and it is very easy to understand. If you are below the ideal line, you are faster than needed, if you are above, you need to speed up. You can see at one glance how much work you have to do in this lesson. It is very good to control yourself.

The Run Up helps you in a different way than the Flap (the board on which the students plan their items of work). On the Flap you see the items or tasks, but they don't have the same effort points. If you put a sticker with a big item plus a sticker with a small item to done, well, that first looks like it was the same effort. On the Run-Up I can see how much work really is done.

Students like the teamwork; 'Teamwork creates team feeling. If everybody in the team has already finished something and you don't, that doesn't feel nice.'

You feel responsible. That is because teams are built on different qualities. Everybody has a purpose.

If you don't do anything, you let your team down.

At first I thought I could do everything at the last minute, but now I prefer to work with the team.

What is important to the students:

If you start with eduScrum please explain very well! If everyone gets it, students can work on their own. And please show us, why it is important to do an assignment. We want an assignment with context that shows us why it is important to learn.

## 8 Student Experiences

### 8.1 *Lars' Student Experience with eduScrum*

eduScrum has influenced my way of working in chemistry lessons, but in a positive way. I think I'm sure to have an advantage through eduScrum. Because of the method, there are better results and each individual makes a personal development. To me, eduScrum's essence is that the work to be performed is more effective and efficient, with cooperation playing a major role, leading to better products and results.

With eduScrum we learn a lot from each other. You get to know other people, you value their talents and you develop new qualities yourself or develop the qualities you already have. There is a continuous exchange of qualities, ideas and opinions. You must use your own abilities and face the qualities you do not have or which are still underdeveloped.

The way of working was very pleasant. First of all, the composition of the teams on the basis of talents and skills is a strong point of eduScrum. Because there is a great diversity in properties one can motivate and learn from one another, this motivates the working attitude and working atmosphere. There is appreciation between the team members. If a team member does not understand anything, then there is the team that can help. The team is thus actively able to help each other and answer each other's questions without a teacher. Students thus work independently, self-studying and with responsibility.

There is a large team accountability, a team interest and mutual involvement. This leads to better cooperation. Of course, there were people who have given more input than other people, but this will be the case in every collaborations and teaming.

You learn to work with people who prefer to see things differently than you do yourself and you learn to take into account other people. Due to the transparency of the process, barriers become transparent, clear and can therefore be easily resolved.

Result-oriented work and active work lead to a favorable and better result. By planning and working in a structured way, a better result can be done in the same period of time.

eduScrum makes a clear change in the completion of lessons and the way in which work is done by our team. We are the boss and owner of our work. Each lesson is different and the aim is different for each team. Because there is a common team interest, the pursuit is often achieved. Every team member will ultimately make a good result; Perform well on the test or a good grade for a report.

To achieve this goal, the team members are strongly dependent on each other. After all, a result must be put down in which everyone has accomplished his activity. Each team member has accountability for the team, which motivates the student.

Students are actively engaged in their work. The moving of the post-its and this gives every time a kick: ‘Yes, some more!’ And ‘Yes, next task!’. The effort that is being provided for the work form ensures good results.

What eduScrum brings to my team members is that they get to know themselves and work better. Their way of learning and work will change. eduScrum gives you more overview, planning and structuring work, and achieving results-oriented. Everyone makes personal development, conscious or unconscious. Everyone will learn from the method. You get appreciation from classmates. You are getting to know new people. You will get to know yourself. You are less dependent on a teacher. You will be helpful. You become responsible. You get discipline. You get a sense of team accountability. You get motivation. And at the end, you get better grades and results.

We become better prepared for the future.

You can learn from each other and be an example. If someone is well in charge, this can inspire a team member to develop himself. There is an exchange of qualities and eduScrum takes care of this. If each person had worked in class, everyone made his own thing, each his own planning, each his own responsibility; then there had been no exchange of qualities, people did not get in touch, people did not appreciate each other and the students would always be dependent on the teacher.

What does eduScrum bring me?

I learn new things and can be an inspiration for other people. With eduScrum I can share all my qualities in the team and in the class. eduScrum will teach students what they can do well. These confirmations ensure that you know yourself better, your qualities and the flaws. This allows you to know where to work on to improve yourself. So you can learn new things to yourself. eduScrum plays a major role in this.

## ***8.2 Marente’s Student Experience with eduScrum***

eduScrum is for me an effective and efficient way of working that is results-oriented. Cooperation is of great importance. Working with this method is very good. I usually don’t like collaborating, I rely on my own ability. eduScrum has changed my viewpoint.



eduScrum lets you learn and contribute a lot, creating a personal development that is positive for the future. You learn about yourself and get to know others well because qualities are discussed. Through team accountability, everyone works better and teaches you to take more account of others. Because you are actively engaged in each lesson, you manage the dust faster and better. This also leads to a good result!

What is fun about eduScrum is that you can work in a team and that it is very cozy. You will get to know your own qualities better, and you will develop them a lot over a period of time. You learn more about yourself, but also from others. Because you work in a team, you can help to explain difficult topics. And you get higher grades!

One disadvantage of eduScrum is that sometimes it's a bit of fun in a team, you are more distracted, and sometimes it is hard to work. You are dependent on your team members with an assignment. So, sometimes you have to push your team members to get them to work so you can achieve the goal per lesson.

eduScrum was of enormous help me for my personal development. It gives me a lot of insight into myself, what I already can and what there is to learn. It shows me, that I can do much more than I thought, which gives me confidence.

With eduScrum you learn a lot about yourself at this young age. Knowing yourself now, you can change a lot, focusing more on improving properties. It's not only a method to deliver a better result. It creates a development that can offer you a good future!

My team members will also get to know themselves better and bring their qualities forward. Their way of working will be changed by my influence and vice versa. They also have developed personal growth that will benefit the future. Unnoticed, they have made a change that they can be proud of.

eduScrum ensures that you always work in an orderly manner, and even if there are obstacles you can eliminate them. By working in a team, I have learned to work together better, my trust in other students has increased as they contribute the right way. eduScrum rewards you for your work and I like that. It gives you the kick to work even harder to achieve the optimal result.

I will also want to work more with eduScrum in the future, in a team that works fine for me. I know that I can rely to my team mates and they to me. Everyone makes their own contribution and I can trust everyone. It is a well-structured way of working and I like it very much, so I know exactly what I need to do to deliver a good result. I am very result oriented and want to get the best out of myself and my team. eduScrum fully subscribes to this!

## 9 Summary

eduScrum is simple, but it is not easy. You cannot do eduScrum halfway. Each part is there for a reason. If one single eduScrum component makes your situation better, it is obviously smart to apply that. Fine. But that does not make your teaching eduScrum yet. You should not seek to adjust eduScrum to your situation because eduScrum, like Scrum, is a system that works like a Swiss clock. Whatever you do and how

you apply it, use all the elements. It is a precarious game. If you want to use parts of eduScrum because it seems useful, please feel free to do so. You just do not gain all the benefits that can be achieved. eduScrum works as a whole and delivers more than the sum of the parts.

Fail early, fail often. Be sure that your first eduScrum project will ‘fail’ in some way. It would be weird if applying this new way of working would immediately seem smooth to you. When you are starting with eduScrum classes, you go on a journey that will probably be new for you. Errors are being made and lessons are being learned thanks to these errors. Trust yourself and be assured that every new project, every sprint, even every day you will feel more comfortable with that new situation. Make your mistakes, forgive yourself and learn. The same applies to your students. Therefore, tell your students that they should be able to trust themselves and that they are allowed to make as many mistakes as possible, the sooner, the better. They learn from it and certainly have the time to adjust. As long as you and your students work together as a team in a trustful atmosphere, everything will work out all right.

## References

- Covey, S. (2006). *The speed of trust*. New York, NY: Free Press.
- Delhij, A., van Solingen, R., & Wijnands, W. (2015). *The eduScrum guide*. Retrieved from [http://eduscrum.nl/en/file/CKFiles/The\\_eduScrum\\_Guide\\_EN\\_1.2.pdf](http://eduscrum.nl/en/file/CKFiles/The_eduScrum_Guide_EN_1.2.pdf).
- Grenning, J. (2002). Planning poker or how to avoid analysis paralysis while release planning. *Hawthorn Woods: Renaissance Software Consulting*, 3, 22–23.
- Lencioni, P. (2002). *The five dysfunctions of a team: A leadership fable*. San Francisco, CA: Jossey-Bass.
- Martin, J. R. (1995). A philosophy of education for the year 2000. *The Phi Delta Kappan*, 76(5), 355–359.
- Moen, R., & Norman, C. (2009, September). *Evolution of the PDCA cycle*. Paper Presented at the Asian Network for Quality Conference, Tokyo. Retrieved from <https://www.westga.edu/~dturner/PDCA.pdf>.
- P21. (2016). Framework for 21st century learning. *Partnership for 21st Century Learning*. Retrieved from [http://www.p21.org/storage/documents/docs/P21\\_framework\\_0816.pdf](http://www.p21.org/storage/documents/docs/P21_framework_0816.pdf).
- Schwaber, K., & Sutherland, J. (2017). *The Scrum guide: The definitive guide to Scrum*. Retrieved from <https://www.scrumguides.org/docs/scrumguide/v2017/2017-Scrum-Guide-US.pdf>.
- Sinek, S. (2009). *Start with why: How great leaders inspire everyone to take action*. New York, NY: Penguin.
- Sutherland, J., & Sutherland, J. J. (2013). *Scrum: The art of doing twice the work in half the time*. New York, NY: Crown Business.

# Getting Agile at School



Paul Magnuson, William Tihen, Nicola Cosgrove and Daniel Patton

**Abstract** Teachers at an international boarding school began experimenting with Scrum as a way to structure self-regulated learning in the context of a class taught in 2013–2014. In the 4 academic years since then, teachers have developed Kanban boards as individual, group, and classroom-wide organizational structures and trialed a number of concepts familiar to practitioners of Scrum, e.g., sprints, burn-down charts, and retrospectives. Working with the support of the school's professional development department, teachers engaged, in their particular classroom contexts, with action research cycles of planning, doing, reflecting, and redoing until arriving, at least for the time being, at ten practices of an Agile mindset for teaching and learning. Each of these will be familiar to educators. The thinking is that small adjustments in multiple practices are not only manageable for teachers, but also more likely to shift educational practice away from the tendency to rely heavily on carrot-and-stick traditions. Informing our practice in school with insights from the Agile revolution in industry is a way of suggesting that many of our current educational practices are in need of an update. Further, incremental change, shared by many, can be a powerful tool to create learning that organizations can be proud of.

**Keywords** Agile learning · Agile in education · Agile mindset  
Project-based learning · School improvement · Student self-regulation

---

P. Magnuson (✉) · N. Cosgrove · D. Patton  
Leysin American School, Leysin, Switzerland  
e-mail: [pmagnuson@las.ch](mailto:pmagnuson@las.ch)

N. Cosgrove  
e-mail: [ncosgrove@las.ch](mailto:ncosgrove@las.ch)

D. Patton  
e-mail: [dpatton@las.ch](mailto:dpatton@las.ch)

W. Tihen  
Garaio, Bern, Switzerland  
e-mail: [william.tihen@garaio.com](mailto:william.tihen@garaio.com)

# 1 Introduction

## 1.1 *The Agile Mindset*

A group of teachers at the Leysin American School have been leveraging the school's professional development program to experiment with different approaches to teaching and learning. What began as a remedy to a struggling project-based curriculum in a class taught in 2013–2014 has grown, often through an approach of ready, fire, aim (e.g., Fullan, 2011), into a cultural shift at the school. By the 2017–2018 school year, several of our teachers were pulling Agile into education, of course with different levels of understanding and differing results. We felt the time had come to make a clearer statement of what we meant when speaking about having an Agile mindset, with the dual goal of creating a shared vision and a professional development process that would help us continue to focus on the type of learning environment we are trying to create.

While Agile in education is relatively new, Agile itself is no longer new at all. Practitioners in fields where Agile grew up speak of “modern Agile” (TechBeacon) and a “post-Agile environment” (Cockburn, 2017). As we travel our own implementation path now in education, those of us pulling Agile into our practice can benefit greatly from their experience. A signatory of the original manifesto, Alistair Cockburn, reminds us with the Heart of Agile (Cockburn, 2014) that, at its core, Agile is a way for teams to get things done. In education, we have plenty of teams (classes, student groups within classes) that are required to get an awful lot done (learning). Cockburn has boiled Agile down to this: collaborate, deliver, reflect, and improve. Simple enough to remember, but deep enough to drive a lot of work and potentially a lot of change. Take the notion of collaboration. Yes, as educators we often create activities for students in pairs or groups. But there are also plenty of admonitions to do one's own work, use of grading curves that pit students against each other, classes taught behind closed doors, and a lack of collaboration between the school and students concerning what is even worth learning in the first place.

As newcomers to Agile, our practices may not be as honed as the Heart of Agile, but we believe that our practices have Agile at their heart. We illustrate them here through a number of experiences in and out of the classroom, real stories that deepen our understanding and let us share our efforts for consideration in your context. Taken together, the vignettes create a sense of the culture that our version of Agile in education can create. Ultimately, our goal is a shift in culture, from an emphasis on teaching, for example, to an emphasis on learning. From an emphasis on teacher as knower to teacher as learner. From an emphasis on satisfying what the teacher, the curriculum, and the school want to satisfying yourself, as the student, as a self-regulated learner.

Agile for us is a mindset. We have no prescription, no method, and no program, though we continue to benefit by learning about prescriptive methods like eduScrum (Delhij, van Solingen, & Wijnands, 2015) and Scrum@Schools (“Manifesto for agile education,” 2015). We learn from the ongoing conversations of why we as a group

of learners (and here we include students and teachers) prefer this over that. (See the Agile manifesto, Beck et al., 2001; the Agile in Education Compass, Robbins, 2016; and the Manifesto for Agile Learning, Scrum@Schools, 2015 for examples of right-shifting thinking.) For example, we prefer a culture that favors student self-regulation over teacher control. We do not claim that student self-regulation is always required, nor do we villainize teacher control. But we strive toward student self-regulation when it is practicable. And maybe sometimes even when it is not.

## ***1.2 The Core of Agile in Education***

These are our values, the core of the mindset we are working toward

- **EXPLORATION**—Exploration and play over tests and perfection;
- **GROWTH**—Growth and rework over assessment reports without corresponding mechanisms to improve identified weaknesses;
- **SELF-REGULATION**—Student-driven reflection and improvement over teacher directives; and
- **LIFE WORTHY LEARNING**—Learning that supports additional learning over detailed course content.

In a quick sentence, we might describe our efforts this way:

As we strive to understand learning and its outcomes better, we look for student and teacher growth through self-regulated, collaborative exploration and play.

## ***1.3 Our Context***

We are pulling Agile into education at the Leysin American School in the Swiss Alps. We are a boarding school with 320 students from approximately 50 countries, in grades 7 through 12. The school is structured in three grade bands: grades 7 and 8, grades 9 and 10, and grades 11 and 12. We write here mostly about the middle school, grades 7 and 8, where the curriculum is most flexible. This is practical for a number of reasons:

- the recent creation of the new middle school signaled to everyone that changes were coming, helping us to manage expectations;
- we opened the new school with only 18 students, or about 5% of the whole school population; and
- in the American education system, grades up through grade 8 are not included on high school transcripts.

These conditions point to a key enabler for us as we pull Agile into education. We have an environment in which we can fail safely (Elia, Lockard, & Ackerbauer, 2017), simply because we do not draw too much scrutiny.

More than a year before we started planning the middle school, and before we started framing our understanding in terms of Agile, we were experimenting with what we felt were common sense notions of working together as students and teachers. When we implemented eduScrum (Delhij et al., 2015) in one class, the conversation continued, but with the addition of terminology familiar to agilists. Along the way we began internalizing an understanding of agility with the help of John Miller's Agile Classrooms (Agile Classrooms, 2018), among others. By the time we started middle school planning, it was clear that the Agile Mindset would inform our teaching methodology. Through a combination of planning, modeling, observing, talking, and writing we have settled on—now in our second year of the middle school—ten interrelated practices. They are:

- EXPLORATION—Exploration over fixed content
- GROWTH MINDSET—Growth over stasis
- TRUST—Self-regulation over teacher control
- TRANSPARENCY—Visibility over obscurity
- ADAPTABILITY—Flexibility over rigidity
- SMALLIFY—Quick, workable iterations and feedback over big plans
- VALUE—Valuable learning over convenient assessments
- COLLABORATION—Working together over competing against
- REDO—Reflection and progress over right and done
- UPLIFT—Problems as opportunity over problems as problems.

It is during the academic exploration classes of the middle school, classes that are only 4–6 weeks long, where Agile practices tend to flourish. There are a few other promising areas, notably the year-long course Physical Education and Health, and some non-course examples, like our faculty process for reflecting on courses during curriculum review and the professional development we engage in to support ongoing implementation. In all cases, the interest and commitment of the individual teacher or teachers is key.

Below we illustrate the ten actionable practices with vignettes from our own experience. At the end of each vignette we have provided some cross-references with other Agile practices that also apply. You be the judge if the overlap suggests we should combine some of our practices. You might also like to wonder, as you read, which practices we may be missing.

The short classes the vignettes refer to are

- DIY Language, in which students construct a new language as an introduction to the study of linguistics;
- Ideal School, in which students design a school and present it in a poster session during an annual school conference;
- Introduction to Engineering, in which students create and pitch designs they create themselves;
- Project Innovate, in which students select a project of personal interest; and
- Robot Gardeners, in which students use Arduino to build gardens in terrariums that can thrive without human attention.

The full-year class in the vignettes is:

- Physical Education and Health.

We hope that sharing our stories helps bring alive the cultural shift we are trying to grow.

## 2 Ten Actionable Practices

### 2.1 *Exploration*

I believe that every human being with a physically normal brain can learn a great deal and can be surprisingly intellectual (Asimov, 1980, p. 19).

Do-It-Yourself Language is a backdoor into the study of linguistics. In groups of eight, students write a dictionary and build a set of grammar rules until they have enough language to put together a skit, which they perform for the class and others as the final project. Our first language was called Blasa, from the root “bla,” which you might guess the students took from the English bla-bla-bla. We have also created Yeuropean and Chuankglish.

This type of project-based learning turns the traditional curriculum inside out. Instead of following a list of topics that we have determined before the class begins, we treat the linguistic topics that arise at the moment they arise. We know we will have good conversations. We just do not know what they are going to be about. When you are exploring you expect to be discovering things.

For example, the students are generally inclined to follow their native language when creating a new system, say, pronouns. English speakers may initially want to replace I, you, he, she, it, we, you all, and they (let alone my, mine, myself ...) with a different word for each pronoun. Referring them to Mandarin provides a logic that is much easier to learn—and therefore a better choice when building an easy language from scratch. The Mandarin pronoun system is so elegant that the conversation arises: are Mandarin speakers more logical in general because their language is more logical? Voilà, we’re discussing Sapir-Whorf and one of the more common questions in sociolinguistics: does language affect thought?

When the students build a number system, they find out that some systems are quite logical (Japanese, Korean, Finnish) and others, well, you have probably heard of how French say 95 (four-twenty-fifteen) and you can ask a Dane why they count as they do. So are speakers of languages with regular number systems better at math? Researchers have studied this. So, students, what did those researchers find ... and if you are interested, why not look into it further?

Students get to choose what parts of language they think they need to create, how that part of language is going to work (grammar) and sound (vocabulary), how it is represented (letters, characters, other?) and so on. Exploration breeds motivation. Learning something just at the point where it has captivated one’s attention is key.

It makes us wonder ... are there other courses that we could be renaming—and re-teaching—as DIY Something-or-Other?

See also TRUST and UPLIFT.

Students in Physical Education and Health choose topics, within teacher prescribed parameters. Incorporating student choice allows them to explore topics in which they are interested. The basic steps used in Physical Education and Health are

- Give the students parameters. Too much choice, i.e., broad or unclear parameters, can lead to paralysis and little or no exploration. For example, allow students to select from racket sports or sports that involve using a ball.
- Allow students to identify what they want to learn within the broad topic. What do they want to know? What puzzles them? What can they go over again if they feel less confident than others? Make these questions the goals for the unit and keep them visible in the room to refer to later. Students can add to them as they like.
- Revisit the goals from time to time. Are students actually focused on them? Are they finding answers? How do they rate their progress? And can they add or take out goals as the unit progresses? Sure!

See also TRANSPARENCY and ADAPTABILITY.

## 2.2 *Growth Mindset*

When you get to the top of a mountain, keep climbing (Kerouac, 1958).

Each year, our research center selects a number of faculty members to be Resident Scholars. These teachers commit themselves to a year-long investigation of their choosing. We support them with a small stipend and possible assistance to present their work at a conference or other professional development event.

A few years ago, a math teacher combined her resident scholarship with a graduate course, setting up a grading system in her algebra class closely aligned to standards based grading, as contrasted with what our entire school was doing at the time, a traditional American 90% A, 80% B style of grading. Drawing on work by O'Connor (2011), she transformed grading in her math class, published her work in our research center's publication, *Spotlight* (Gorasia, 2015), and then unfortunately accepted a position elsewhere. Her work is still very alive at our school, however, as it was her model we adopted when creating the middle school.

Key to our middle school grading system is the notion of “not yet.” On any assignment, project, or test, a student receives a 4 (you know this so well you could teach it), a 3 (you know this well enough to move on), or a 2 (you do not know this yet). After demonstrating sufficient effort to learn the material, students may reassess to bring a 2 up to a 3 or 4.

While not all teachers immediately adopted the system and we continue to discuss exactly how it should be implemented, we are seeing certain effects. The summary here is that we're supporting a growth mindset over a fixed mindset (Dweck, 2016).



The interesting supporting evidence is that students are never “let off the hook” by simply receiving a bad grade and moving on. Instead, they must learn the material. Further, teachers have an incentive to make good use of formative assessments to see if students are ready for a summative assessment, since teachers know that assessing for a grade before students understand the material leads to review and redo for some students while others are ready to move on. This puts pressure on the curriculum to adapt, expanding or contracting according to how much time students need to get to a 3 or a 4.

Further still, we as teachers often bemoan the resistance students have to risk taking, yet some of our most common grading systems reinforce their aversion. If the result of taking a risk is a low grade, figured in an overall average, for evermore on the transcript, why take the risk? Assessment can easily hinder a growth mindset. We think we’re getting around that, as many schools have done before us, by re-examining the method by which we assign grades to student learning.

See REDO and ADAPTABILITY.

### 2.3 *Trust*

My appeal is to observation — observation that each of you must make by [yourself].—Charles Sanders Peirce (Turrisi, 2007, p. 140)

We want to place lots of trust in our middle school students to further their ability to self-regulate. So, we experimented with a new way for the middle school director to observe classes; namely, by asking the teacher not to be present. Certainly then the students will have room to demonstrate the degree to which they are self-regulating! The director takes notes on how well the students carry on without the teacher, then talks through the notes with the teacher when the lesson is over.

After 5–6 weeks of experimenting with Agile practices, the Physical Education and Health teacher invited the director to observe a teacherless lesson on badminton. She was naturally a bit nervous about how it would go. Would students use the tools they had and plan a lesson that demonstrated good learning? Might it be a complete 90-min fail?

The tools students had practiced with and that were available during the observation were

- A visual task board (e.g., a Kanban board);
- A reflection sheet for planning future lessons;
- Paper stars with questions that allowed each student to check for understanding and earn points;
- A folder about badminton that included ideas, links, and drills to help if students were stuck;
- Assigned roles for various parts of the lesson; and
- Previous teacherless moments—on several occasions, the teacher had practiced handing over control to students to observe and debrief the dynamic of the class.

**Table 1** Notes from a teacherless observation

Time	Student actions as described by the observer Individual students are referred to by first initial
08.11	All set up. R asks J to tell everyone what they are going to be doing. R moves a sticky into the DOING column. They start an exercise
08.15	First exercise ends. J leads stretching. R updates Kanban board, moving the first exercise to DONE and the next activity to DOING
08.16	J writes on the board. Everyone else gathers around. They are figuring out a tournament structure. Nobody tries to talk to me, to ask me what to do, to wonder if I'm in charge
08.18	V fills in who will play whom. A is a little aloof. V coaxes everyone to come start. The first players take the court
The students play the tournament ...	
08.41	Tournament over. J always wins, says V. J borrows my computer and starts video. One of the students updates the Kanban board
08.46	J says "let's practice" and everyone goes back on the courts. There's a mix of practicing what was on the video and maybe just playing for fun. The students finish practicing and set up another tournament
08.52	All students take to the courts. I'm impressed by the quick transition times. The equipment got set up quickly, the time to switch from practice to tournament and the other transitions are all pretty quick
08.53	Tournament 2 is underway. 4 players, 2 scorekeepers, and A sitting on the bouldering pad in the corner
The tournament finishes ...	
09.08	Students pull questions (on stars) and ask each other randomly. All participate except A. He is holding a question, but not asking or answering
09.10	Now A asks J a question. Perhaps he is just more patient than everyone. Students continue to ask each other questions and fill in points on the board for each other
9.12	C says something in Chinese to J. B says "English please." C switches and explains with gestures, in English
09.13	Nice laughter, explanation, self-organizing pairs. W spells the words "drop shot" for J
09.14	Students determine they have more time and W and V set up new tournament. R and J clean up the star questions and the students start to play
	End of observation

The results were much better than she had hoped. The students were able to set up, lead, and organize themselves, practicing badminton skills that they had previously highlighted for practice and improvement. In Table 1, snippets from the observation notes tell the story of how this particular lesson unfolded.

It takes a good degree of trust to turn the class over to students, whether for ten minutes of small group work time or a 90-min PE class. What if they waste the time? What if they get off-task? We counter by asking when students are going to learn to self-regulate if they do not practice doing self-regulation, which will necessarily

include off-task time. In a very real sense, what we see in a teacherless observation is student learning, not teacher teaching. Or to dig a bit deeper, we see in the student self-organization the fruits of the teaching that lead up to the self-regulation observed in this lesson.

See also GROWTH MINDSET and TRANSPARENCY.

## 2.4 *Transparency*

Honesty and transparency make you vulnerable. Be honest and transparent anyway (Keith, 2002).

A few years ago, John Miller trained us in the use of his Learning Canvas. Last year, Willy Wijnands of eduScrum trained us using his Flap. Internally we have been calling it a Kanban board, though it might be a Scrum board. The names don't matter so much—but the visibility that a Kanban board supplies, whatever one chooses to call it, does.

A simple three-column Do, Doing, Done display is an easy and practical first step toward shifting the culture to a more visible workflow. We used it for planning the middle school (in Trello) and during the first year in the daily homeroom (on a poster). Most middle school teachers transferred the homeroom example to their classrooms, gradually customizing to create their own versions. Some teachers used labeled magnets on the whiteboard, some used laminated cards. Other classes began using smaller Kanban boards for small groups or individuals, usually inside a standard manila folder. Students working on individual or group projects (e.g., coding, touch typing, Ideal School) set the Kanboard board next to them so that they—and the teacher—could quickly monitor progress.

It is a small thing, but we have found that transparency tends to create good, and often unexpected, developments. For example, the teacherless observation described in TRUST above is a direct result of the Kanban board in an English class. The director filled in for the English teacher one afternoon. Her directions stated that the students would start the lesson with a stand up at the Kanban board. It felt so promising that the director told the students to go ahead and run the whole class while he watched. The teacherless observation was born. Other teachers, trying to keep track of individual progress in loosely structured classes, took pictures of individual or small group Kanban boards at the beginning of the class to make sure that students made progress by the end of class. While perhaps diverging a bit from our TRUST practice, it is a pragmatic option that wasn't available before Kanban boards. Finally, use of Kanban boards has begun extending beyond the classroom, to student planning in the resident halls, to faculty professional development sessions, and to the dean of students and admissions offices (which now use Trello).

It's a simple tool, yet it seems to be a reliable driver of the cultural shift we believe we have started.

See also COLLABORATION.

## 2.5 *Adaptability*

The acquisition of knowledge (i.e. the process) is more important than soon outdated content (Quinton, 2010, in Weichart, 2013, p. 43).

In our work with whole school curriculum, we recently solved one of those problems that seems easy on the surface but somehow manages to get everyone stuck, repeatedly, until it does not seem there is a way to get unstuck. A few of us are calling our adopted approach “lean curriculum.” Here’s why: More often than not, in what Sutherland (2014) refers to, albeit in other contexts, as a “tall stack of futility” (p. 11), curriculum is presented as a pre-established plan for the entire academic year, usually in every subject, for every class, and in great quantity. And then instruction starts. But as Apple and Jungck (1991, in Fullan & Hargreaves, 1996) point out, “dumping curriculum packages on teachers ... tends to make them deskilled and dependent” (pp. 101–102).

Our lean curriculum places an emphasis on the ongoing, never-ending process of comparing the curriculum plan with what is actually being taught. The two naturally diverge; that is not a problem, it is an expectation. Teachers take the curriculum for a guide and then do their thing, teaching to their strengths, expanding where there is interest, cutting short where there is none, slowing or speeding the pace to make content and available time come out even, and so on. They are, in short, “responding to change over following a plan” (Beck et al., 2001).

Our process requires a lean curriculum, literally four pages for a class for the academic year, which is made available for comments to all faculty members of the school. Every teacher and staff member can leave suggestions for any course. Then, at a minimum, once a semester all the members of the academic department resolve the comments on the curriculum document, in an effort to resolve the divergence between the curriculum plan and what is taught:

1. Is what the teacher taught more on target (in respect to student interest, preparation for future courses, parent expectations, etc.) than what the curriculum suggested? Then change the curriculum to match what is taught.
2. Is what the curriculum suggested more on target than what the teacher taught? Then reinforce with the teacher what needs to be taught, why this is so, and provide assistance if something is blocking that teacher from teaching the agreed-upon curriculum.

There may be a comparison here between project planning using a Gantt chart, for example, and the cyclic iterations of agile acting and planning. We are aiming for the latter.

The lean curriculum is nimble and allows ongoing review in the short periods of time available to teachers. And there may be other benefits. Bankston (2017) describes a vocational school in Rotterdam that encourages a 70–30 split of planned and unplanned curriculum so that there is room in the backlog—the collection of possible lessons—for professors to introduce material based on student interest, student need in a particular year, and so on. In other words, while we may as curriculum

writers and administrators feel we have done a good job if we have each day planned, in detail, perhaps we should be intentionally making sure that three out of ten days (in the Rotterdam example) are not planned, at least not until they need to be planned. Slack time may in fact be a necessary enabling factor for many of our actionable practices. Exploration and play may thrive better in unstructured time. Further, as a general warning familiar to the Agile community: beware of the big plan!

See also TRUST, VALUE, and SMALLIFY.

## 2.6 *Smallify*

Agile methods derive much of their agility by relying on the tacit knowledge embodied in the team, rather than writing the knowledge down in plans (Boehm, 2002, p. 66).

Staff turnover is a constant confounding variable when implementing change. International schools can be particularly good at turning over staff, to a large degree because international teachers are a self-selected group who love to travel. For this reason no one was surprised when, already in our second year, two teachers new to the middle school and to the Agile Mindset joined the team. During orientation week they asked: “When is the training for Agile?” We fumbled around for an answer, saying that we would get to it later or something equally unsatisfactory. But in reality, that is exactly what we are doing. We train as the year unfolds, in our classes, and through our conversations as topics arise. In education this could be referred to as classroom-embedded and ongoing—two hallmarks of solid professional development. And each interaction is short. We might ask: Did you see what just happened? The students directed the next move because you made the process transparent. Or: What do you think if we do this during the next activity—how might it lead to students asking deeper questions?

The same is true for our students, too. We will always be faced with continuous implementation of the Agile Mindset in our middle school because we will always be working with students new to our school. We will never be able to “train” them in agility during the orientation week. We will do it along the way, modeling teaching and learning that deliberately creates short feedback cycles, celebrates little bets, and provides multiple opportunities to fail safely, because any individual fail is small enough to climb over, or climb up on, to reach higher.

See also VALUE and UPLIFT.

## 2.7 *Value*

Very few schools teach students how to create knowledge ... Instead, students are taught that knowledge is static and complete, and they become experts at consuming knowledge rather than producing knowledge.—Keith Sawyer (Sims, 2011, p. 160).

For 4–6 weeks each year, we ask students to come to a class called Project Innovate, in which they must create the curriculum. “What would you enjoy working on?” we ask them. “What do you want to learn?” And then we have to trust that, confronted with this golden opportunity, they can handle it. Often they cannot, particularly at first.

It would be hard to assess in this class. Students are being given the chance to choose something to learn, pursue it, go deep, share it with others . . . and they may fail. We have seen students wander from project to project. We have seen many students pick something far too complex, at least initially. We have seen others stick with a project and create an excellent end product, while others do two or three reasonable projects, and still others flounder. We believe the experience is important—a gut check of sorts that allows the students to experience firsthand how self-regulated they are, or aren’t, at this stage in their schooling. It also affords the opportunity for those who can’t—at least those who can’t *yet*—to observe those who can, without a penalty.

Now, what if we were to assess them? Could we allow casting about, floundering, and, well, failure? We might be accused of wasting valuable school time during Project Innovate, throwing away those 15 class meetings, or 15 h, for some students. On the other hand, what if those 15 h demonstrate to the students who need to learn it most, both intellectually and emotionally, that they are going to need much more practice learning how to self-regulate? That it is not easy? That there is a gap to cross between where they are now and where they will need to be in order to self-regulate well?

See also TRUST and GROWTH.

At the end of our Introduction to Engineering course, we set up a role play, complete with a small story, as context. Students must introduce the company they have created in pairs or threesomes, along with their company website and their company product, to students, teachers, and administrators at the end of the course. Everyone in the audience is given play money and turned into angel investors, who are looking for excellent products they can invest in. But that’s at the very end of the class. What leads up to the demo day is also important.

As part of the curriculum, students must add to their project at least one feature suggested by another group of students (see COLLABORATION below). This gives the students practice sharing and getting ideas from each other throughout the class. Then at the beginning of the last week of class there is a practice presentation, to get ready for the final presentation to teachers and administrators playing the role of angel investors. The lead times gives students time to adapt and redo their demos based on peer feedback. Inspired by each other, most of the student groups improved their presentations and most were inspired to update their web pages as well. None of this was required by the teacher, but students realized the value of their peers’ ideas and were self-motivated to spend additional time in order to make their presentations and web pages significantly better.

See also TRUST and COLLABORATION.

## 2.8 Collaboration

Coming together is a beginning; keeping together is progress; working together is success.—Henry Ford (Collins, 2007, p. 8)

Setting up work in a scrum-like fashion ensures a certain amount of collaboration among our students, as well as some of the pain that comes as groups of people, particularly groups of 13-year olds, figure out how to team up and work well together. The Introduction to Engineering class required students, in pairs or threesomes, to develop a product and share their work online at GitHub. The teacher was consistently firm with students that, when they were stuck, they needed to (1) consult a friend, (2) search online, and only then (3) ask the teacher. The class Kanban board had a column for Stuck (between Doing and Done) with exactly those three levels. At first students did what their prior schooling has trained them to do. They got stuck and shot their hands into the air while simultaneously calling “teacher!” Their expressions were fairly incredulous when the teacher declined to answer questions that students had not first tried to answer themselves. It certainly did create collaboration, however, and a sense that expertise in class wasn’t only embodied in the teacher.

As the class worked toward final presentations, one acceptance criterion for the project was that every group had to credit at least one person from another group for assistance along the way. Just the opposite of a classroom worried about cheating, this classroom was all about sharing to learn. The teacher strives to make the assignments too hard for one individual student but doable by four-fifths of the students, given that they are helping each other.

Over the course of our first year, student presentations gained an important role in the cycle of a project or a course. At the end of the first semester, with no final tests to fill the last days before the holiday, teachers chose instead to help students showcase their learning for the semester. Teachers and students collaborated across classes to develop student displays of work, complete with demos, recordings, and other exhibits of their work. On exam day morning the students prepared the room and their presentations, so that on exam day afternoon they could present their learning to the rest of us. It was successful enough that during Family Week, when parents visit the boarding school classes during the winter term, the students chose to put on a similar exhibition for the visiting parents. Together we had created a culture of collaboration. In good positive feedback loop style, other classes, like DIY Language, came to rely on collaborative final projects instead of exams. We didn’t know it would happen from the start, but the Agile Mindset of classes like Introduction to Engineering has turned some interesting possibilities into reality.

See also VALUE and COLLABORATION.

## 2.9 Redo

Iff ure maeking mistakes iht meenz ure owt thair dewing sumthing. - Kneal Gaemin (Sigmon, 2015)

We challenged students to create a climate controlled “robot gardener” that would use microcontrollers (brains), sensors (light, temperature, and soil moisture), and actuators (LED lights, fans, and water pumps) to keep plants healthy over the holiday break, when the students are on vacation. Robotic design projects are a nice fit for employing the Agile Mindset since they can be broken down into small systems, there are many chances for iterations along the way, and the ultimate answer is unknown. There are lots of different ways to build a robot that can keep plants alive.

Building a robot gardener is a large enough project that smallifying (Sims, 2011) is quite important. Early on we ask students to think about the completed project as a series of features. Once they have identified and made a preliminary design of the different features working together, we ask them to choose one feature, for example light control, to start with, build, and test. Students need to figure out how to build the feature and how to write a program that does what it needs to do. In order to scaffold this process for beginners, we provide a circuit diagram and some sample code. Once they have the feature working, we ask students to change the code to reflect a specific set of plant needs. With each subsequent feature (temperature control and eventually moisture control), we offer less scaffolding, but they follow the same engineering and design cycle.

Once students feel confident about their small feature ability to work, we “break” it. We do this in different ways, depending on the student’s level of understanding. For beginners, we may simply move the sensor to a different pin or change the code so that it looks for the sensor at a different pin. For more advanced students, we experiment with different versions of sensors and actuators or different ways of integrating the feature into the larger project. The idea is to highlight the “bugs” of the design and get kids thinking.

We test for understanding by having students fix the break in a novel way. If we change the pin position, we require students to change the code to allow for the new pin position—not just move the connection back to where it was. If students choose to use one sensor or component over another or decide to change the way the features work together in the final project, they need to explain why. They do this by showing and explaining their work to student colleagues and the teachers.

We learned that students learn best by making mistakes. For example, if a student does not wire a pump to fail in the off position, a failure (say a loose wire) can cause their robot gardener pumps to water continuously, flooding the plant (which is not good for plant health!) and causing a watery mess. They do not learn this concept as dramatically when working with the light and the temperature features, which can also fail unsafely, but the same principle of learning from mistakes applies. In addition, we learned that the process of purposely trying to break the project and creating fixes to these bugs has not only generated “aha” moments, but it is also a very fast way to know if students “get it.”



In any case, the formative assessment in Robot Gardeners is real and generally pretty obvious—things either work or they do not. Students learn quickly what they need to redo and they learn that redoing needs to repeat itself until the problem is solved. As Sims (2011) puts it, students are learning the art, like entrepreneurs, of “failing forward” (p. 53). Merely redoing is not enough. A redone connection to the water pump that still fills the classroom floor with water again the next day is definitely not done. You have to fail into success.

See also SMALLIFY and EXPLORATION.

## 2.10 *Uplift*

Make people awesome (Industrial Logic, 2016).

Uplifting creates safety, in the context of an engineering mindset, when students feel safe to choose their own direction, designs, and peers to work with and when students are accepting of mistakes and missteps along the way. Here are two experiences that describe what we mean.

A group of three boys was designing a self-powered car. Their goal was to make it move two meters without a shove. They were using plastic soda bottles for the chassis, a motor hooked to a fan, and plastic bottle lids for wheels. One day they were arguing loudly. The teacher joined their group and together they looked at the drawing of their design and their task distribution. The teacher asked what each student was responsible to build. They each stated their role. Then the teacher asked why they were arguing. They explained that their parts did not integrate and each student wanted the other students to change their design. The teacher asked why they were not building from the design sketch. It turns out they had rushed the design (because they did not realize its importance), so their sketch was vague and they did not agree on the specifics. So of course, the parts they designed were not fitting together.

Before discussing what it would take to agree on a design, the teacher backed up and facilitated a discussion about how they could work together. After a good 15 min of discussion they were able to talk about making a new design.

By the end of the unit they had built a car that worked and was voted by their peers as the most successful project. When the group cited one thing they had learned (the review was deliberately vague, but the teacher was expecting a technical answer), the group said that the most important thing they learned was how to work together. The teacher reported afterward that he was sure if he had punished them for arguing and disrupting the class, they never would have learned this lifeworthy (Perkins, 2016) lesson.

In another group, one of the more focused students was taking a very long break. (Students were allowed to choose themselves when they took a break and for how long.) When the teacher asked him if he was ready to get back to work, the student said he was stuck and did not know what to do. So the teacher engaged him in conversation.

The student explained that he was building a rubber band car but that it would only travel about 20 cm. He had demoed the car and found that the rubber band, which he had glued to the car, was actually making it stop. After some experimentation together, they decided he needed to cut off the rubber band. He reused the place where it had been glued by converting it into a hook. He was unstuck and able to get back to work. It took him awhile to feel it was safe to ask for help, and it took him awhile to feel comfortable about making mistakes. But after that experience he was much more exploratory and interested.

Like the arguing boys from the other group, turning this boy's overly long break into a discipline issue would not have accomplished much at all, while exploring the problem in a friendly way—without giving him the answer—uplifted him and built his confidence.

See also EXPLORATION, REDO, and TRUST.

### 3 Conclusion

Conclusion is a misnomer. Continuation is more accurate. The worth of the Agile mindset as a thinking and working culture lies in its application. Similarly, its application is what continues to create the mindset. Explore, adapt, redo, grow ... we recognize change as normal and healthy.

We hesitate to say that any particular actionable practice is more important than another. They are all, in fact, important and they are all interdependent. And of course there may be more actionable practices that we have missed here, practices that we will discover as we continue creating a shift in culture. We wonder about SLACK, for example. Should unplanned space in learning be emphasized more? What about CHOICE? Are either of these implied in our existing practices, and is merely being implied good enough? We expect to find out over time, depending of course on how our Agile mindset and culture move forward.

Among our existing actionable practices, we do feel that EXPLORATION is really only possible if time is made available to do it (there is the notion of SLACK again) and if our learning environment is truly infused with TRUST and a commitment to UPLIFT. Without these, any practice of Agile in education might give unintentional rise to stress, since the necessary conditions to support exploration, with all its quirky twists and turns, successes and failures, may be registered by students, teachers, and administrators as mistakes or wasted time instead of positive learning experiences. Working on just one or two of these practices should be beneficial, as each individual practice contains some element of good educational practice. But the ideal is to work on many of them, over time, exploring the manners in which they reinforce each other and to witness how that transforms teaching and learning in the classroom and beyond.

Recently we have begun training using the actionable practices we have outlined, collecting stories in the form of these short vignettes as the teachers reflect on the teaching and learning experiences they are creating in their own contexts. We ask

teachers to focus on two actionable practices at a time, while being mindful of their interconnectedness. A teacher, for example, might choose to focus on ADAPTABILITY and UPLIFT, because she has identified the former as difficult and the latter as a practice she excels at. She can both address a weakness and strengthen an asset. Over time strengths and weaknesses will shift and new actionable practices will be identified for improvement, allowing teachers to choose how to adapt and grow as they add self-regulation to their teaching style. In doing so, the teachers are practicing what we are asking students to do, and the students have multiple good models to learn from. The stories of Agile practices that we collect along the way, provided by teachers right from their classroom experience, will inform teaching and learning and the future iterations of our Agile culture.

We are not concluding, we are just beginning.

## References

- Agile Classrooms*. (2018). Retrieved March 29, 2018, from <https://www.agileclassrooms.com/>.
- Beck, K., Beedle, M., Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., ... Thomas, D. (2001). *Manifesto for agile software development*. Retrieved October 25, 2017, from <http://www.agilemanifesto.org/>.
- Asimov, I. (1980, January 21). A cult of ignorance. *Newsweek*, 38, 19.
- Bankston, A. (2017, July 24). Agile and lean outside of IT. *Agile Uprising*. Podcast retrieved from <https://agileuprising.libsyn.com/agile-and-lean-outside-of-it-featuring-arden-bankston>.
- Boehm, B. (2002, August 7). Get ready for agile methods, with care. *Computer*, 35(1).
- Cockburn, A. (2017, August 10). Becoming a meme and agile 2.0. Solutions IQ. Podcast retrieved from <https://www.solutionsiq.com/resource/agile-amped-podcast/becoming-a-meme-agile-2-0-alistair-cockburn/>.
- Cockburn, A. (2014, March 24). *The heart of agile*. Retrieved October 25, 2017, from [www.heartofagile.com](http://www.heartofagile.com).
- Collins, T. (2007). *The legendary model T Ford: The ultimate history of America's first great automobile*. Iola, WI: Krause Publications.
- Delhij, A., van Solingen, R., & Wijnands, W. (2015). *The eduScrum guide*.
- Dweck, C. (2016). *Mindset: The new psychology of success*. New York: Ballantine Books.
- Elia, P., Lockard, R., & Ackerbauer, M. (2017, September 16). Modern agile: Experiment and learn rapidly. *Agile Uprising*. Podcast Retrieved from <http://podcast.agileuprising.com/modern-agile-experiment-and-learn-rapidly/>.
- Fullan, M. (2011). *Change leader*. Hoboken, NJ: Jossey-Bass.
- Fullan, M., & Hargreaves, A. (1996). *What's worth fighting for in your school?* New York, NY: Teachers College Press.
- Gorasia, V. (2015). Assessment using technology in mathematics. *Spotlight: Technology in Education*, 1(1), 12–13.
- Industrial Logic. (2016). Retrieved October 25, 2017, from <https://www.industriallogic.com/>.
- Keith, K. (2002). *The paradoxical commandments: Finding personal meaning in a crazy world*. New York: G. P. Putnam's Sons.
- Kerouac, J. (1958). *Dharma bums*. Google books. Retrieved October 25, 2017, from [https://books.google.ch/books?id=x-HKLD96EkC&pg=PT106&dq=when+you+get+to+the+top+of+the+mountain+keep+climbing+hl=en&sa=X&ved=0ahUKEwjwuu\\_QnIvXAhVEbRQKHcGCmwwQ6AEIJTAA#v=onepage&q=when%20you%20get%20to%20the%20top%20of%20the%20mountain%20keep%20climbing&f=false](https://books.google.ch/books?id=x-HKLD96EkC&pg=PT106&dq=when+you+get+to+the+top+of+the+mountain+keep+climbing+hl=en&sa=X&ved=0ahUKEwjwuu_QnIvXAhVEbRQKHcGCmwwQ6AEIJTAA#v=onepage&q=when%20you%20get%20to%20the%20top%20of%20the%20mountain%20keep%20climbing&f=false).

- Manifesto for Agile Education. (n.d.). *Scrum at school*. Retrieved October 25, 2017, from <http://www.scrumatschool.nl/component/content/article?id=124&Itemid=137>.
- O'Connor, K. (2011). *A repair kit for grading: 15 fixes for broken grades*. Boston, MA: Pearson Education.
- Perkins, D. (2016, March). Lifeworthy learning. *Educational Leadership*, 73(6), 12–17.
- Robbins, L. (2016, Fall). New directions. *AgileVox*, 1(2), 48–56.
- Sims, P. (2011). *Little bets: How breakthrough ideas emerge from small discoveries*. New York: Simon & Schuster.
- Scrum@School. (2015). Retrieved March 29, 2018, from <https://www.scrumatschool.nl/>.
- Sigmon, D. (2015, August 31). 10 motivational quotes for your classroom. *Edutopia*. Retrieved October 25, 2017, from <https://www.edutopia.org/discussion/10-motivational-posters-your-classroom>.
- Sutherland, J. (2014). *Scrum: The art of doing twice the work in half the time*. New York: Crown Business.
- TechBeacon. *Modern agile and the heart of agile: A new focus for agile development*. Retrieved October 23, 2017, from <https://techbeacon.com/modern-agile-heart-agile-new-focus-agile-development>.
- Turrisi, P. A. (Ed.). (2007). *Pragmatism as a principle of modern and right thinking: The 1903 Harvard lectures on pragmatism*. Albany, NY: State University of New York Press.
- Weichart, G. (2013). The learning environment as a chaotic and complex adaptive system: E-learning support for thriving. *Systems: Connecting Matter, Life, Culture and Technology*, 1(1). Retrieved October 22, 2017, from [www.systems-journal.eu](http://www.systems-journal.eu).

# Bringing the Benefits of Agile Techniques Inside the Classroom: A Practical Guide



Ilenia Fronza, Nabil El Ioini, Claus Pahl and Luis Corral

**Abstract** Besides professional programmers, many “end-user programmers” write code in their daily life. Given that so much of end-user-created software suffers from quality problems, Software Engineering (SE) is no longer solely applicable to the professional context: a clear computational processing can be useful in everyday life. While the expansion of programming skills acquisition initiatives in K-12 (i.e., primary and secondary schools) has contributed to improving learners’ coding ability, there have not been many studies devoted to the teaching/learning of SE concepts. In this chapter, we focus on understanding how it is possible to bring the benefits of Agile techniques inside the classroom. Moreover, our goal is to show how each selected practice (such as user stories and pair programming) can be leveraged or adapted to the educational context; to this end, tools already adopted in schools are considered as possible substitutes of professional ones.

**Keywords** K-12 · Agile techniques · XP practices · Classroom · Toolbox

## 1 End-User Software Engineering in K-12: Introduction

Computer programming is becoming a pervasive practice, almost as much as computer use; therefore, the gap between software users and developers is quickly narrowing down (Ye & Fischer, 2007). End-user programming has empowered millions

---

I. Fronza (✉) · N. El Ioini · C. Pahl  
Free University of Bozen-Bolzano, Piazza Domenicani 3, 39100 Bolzano, Italy  
e-mail: [ilenia.fronza@unibz.it](mailto:ilenia.fronza@unibz.it)

N. El Ioini  
e-mail: [nabil.elioini@unibz.it](mailto:nabil.elioini@unibz.it)

C. Pahl  
e-mail: [claus.pahl@unibz.it](mailto:claus.pahl@unibz.it)

L. Corral  
Monterrey Institute of Technology and Higher Education, E. Gonzalez 500,  
76130 Queretaro, Mexico  
e-mail: [lrcorralv@itesm.mx](mailto:lrcorralv@itesm.mx)

© Springer Nature Singapore Pte Ltd. 2019  
D. Parsons and K. MacCallum (eds.), *Agile and Lean Concepts for Teaching and Learning*, [https://doi.org/10.1007/978-981-13-2751-3\\_7](https://doi.org/10.1007/978-981-13-2751-3_7)

of end-users to create their own software: more and more people are not only using software but also taking part to the software development process to widely varying degrees to solve different types of problems. Unfortunately, there is a downside: errors are pervasive in end-users-created software (Burnett, 2009). End-User Software Engineering (EUSE) aims to increase the quality of end-user-created software by looking beyond the “create” part of software development, which is already well supported, to the rest of the software lifecycle (Burnett, 2009).

A key difference between EUSE and traditional software engineering (SE) is that EUSE focuses on “end-users”. In today’s EUSE literature, the definition of end-user developers differs from paper to paper (Burnett & Myers, 2014). Some researchers consider people who have neither experience nor software-engineering-oriented interests; others use it to mean anyone using a particular programming environment that targets end-users; and others use it to mean people who can program in a specific application area only. Ye and Fischer (2007) proposed a spectrum of software-related activities: at the right end of the spectrum are those that develop software systems professionally, while at the left side are those users who just use software systems to accomplish their daily tasks (the so-called “pure end-users”).

As a *desiloing alternative*, Burnett and Myers (2014) suggested defining end-users based on *intent*, to focus directly on the end-user developers’ motivations and values in that particular software development task, which in turn can affect their SE processes (Burnett & Myers, 2014).

Chimalakonda and Nori (2013) identified the following two main categories of challenges that need to be solved when teaching SE to end-users

- *End-users* concerns: (i) end-users do not see the value of learning the principles of SE; (ii) end-users generally program because they want to simplify or automate their own tasks and not to learn SE; (iii) learning SE requires extra time and effort.
- *Instructional design* concerns: (i) end-users focus on domain-specific goals and not on SE; (ii) SE didactics must be adapted to diversified audiences with different backgrounds and needs; (iii) there is a wide variety of processes in SE, but which of them to use in teaching is not clear; (iv) while there are many resources and tools on the web for learning SE, they are not directly usable for end-users.

Thanks also to the strong emphasis being led by the media, coding has been included in many teaching curricula. Therefore, in recent years, the most comprehensive didactic initiatives for end-users have been associated with K-12 (Monteiro, de Castro Salgado, Mota, Sampaio, & de Souza, 2017), also in non-vocational schools (Fronza et al., 2014). In K-12, some students have the intent to learn to program, which means that they are willing to move from the left- to the right end of the spectrum of software-related activities (Ye & Fischer, 2007). Other students can be positioned in the middle of the spectrum: they have certain software development skills, but just develop software with the intent to solve specific problems (Costabile, Mussio, Parasiliti Provenza, & Piccinno, 2008).

Because of the pervasiveness of software in the labor market, it is indeed of paramount importance to equip K-12 students with the necessary means to improve software quality: they will probably be end-users also in their future career, and at

that point producing dependable software could be crucial. Moreover, the benefits of a SE approach can be extended well beyond the engineering field: for example, the ability to work iteratively and incrementally can also be employed in other disciplines and in daily life as well.

In this chapter, we focus on understanding how it is possible to bring the benefits of Agile techniques to K-12 education. To this end, we first describe a toolbox of practices, by showing how they can be leveraged or adapted to the educational context; in particular, tools already adopted in schools are considered as possible substitutes of the professional ones. Then, we detail a selection strategy, to support teachers in selecting the right practice from this toolbox, based on different characteristics of their teaching activities. There are limited examples of Agile training in K-12, and most of these studies have focused on a limited number of practices applied in isolation. The goal of the proposed toolkit is then to leverage a synergistic use of practices.

## 2 End-User Software Engineering in K-12: State of the Art

There are currently only a few studies that explicitly address the need for teaching SE to end-users. An overview of these studies can be found in Monteiro et al. (2017).

Umarji, Pohl, Seaman, Koru, and Liu (2008) conducted a survey in the context of bioinformatics, which focused on the software engineering principles that should be learned by end-users who develop software in the bioinformatics domain. According to the authors, most of the bioinformatics students do not receive formal software engineering training, and usually learn programming principles through self-teaching. As a conclusion, the authors suggested to include software engineering principles in bioinformatics education.

Gross, Herstand, Hodges, and Kelleher (2010) described a code reuse tool for the Looking Glass IDE, which aimed at showing the importance of reuse to middle school students who are learning programming. Using this tool, students can reuse parts of programs written by others, even without understanding completely how that code works.

Some research on Agile in K-12 has recently been performed. Meerbaum-Salany and Hazzan (2010) presented an Agile mentoring methodology for high schools. In 2015, Fronza, El Ioini, and Corral (2015a) implemented a computational thinking training course in which they leveraged the software development phases (i.e., feasibility, analysis, design, development, testing, and integration) to foster specific computational thinking skills. The same group of researchers, in 2017, focused on the instructional design concerns in teaching software engineering to end-users (see Sect. 1), by proposing a framework in which a set of Agile practices were adapted to the middle school context in order to teach computational thinking (Fronza, Ioini, & Corral, 2017).

In 2016, Kastl, Kiesmüller, and Romeike (2016) achieved greater flexibility in software development projects in three secondary schools in Germany, by applying Agile methods.

Monteiro et al. (2017) analyzed how the technology used in a Brazilian CTA program prefigured elements of software engineering in the participants' programs created with AgentSheets.

To guarantee that SE will enter the K-12 environment, a set of how-to's is needed for teachers, so that they will be able to apply SE practices in their classrooms.

### 3 Bringing Agile to K-12 Education

In the first part of this section, we detail why Agile practices fit the K-12 environment, based on the end-user characteristics and work habits in this specific context. Then we explain why, among the possible Agile methodologies, eXtreme Programming (XP) represents a good candidate for K-12 students.

#### 3.1 *Why Agile?*

One of the main challenges in EUSE is to understand which SE process to use in teaching, among the wide variety of possibilities (Chimalakonda & Nori, 2013). With respect to this goal, the EUSE approach is to respect end-users' real intentions and work habits, without advocating to transform end-users into software engineers (Burnett, 2009).

Under a plan-driven development model, once the requirements have been fully specified, the project continues through the design, coding, testing and integration phases, finally leading to deployment of the finished product. In a professional environment, this structure facilitates the creation of contracts, as the product definition is stable. In K-12, a plan-driven approach could then encourage a continuing focus on the product while, according to EUSE goals, students should also learn how to better organize the development process (Steghöfer et al., 2016). Moreover, one of the well-known disadvantages of plan-driven development models is that once an application is in the testing stage, it is very difficult to go back and change something that was not well-thought out in the concept stage. Also, no working software is produced until late during the life cycle. Therefore, using a plan-driven process in K-12 could easily lead to having students with non-working products, which might actually be used to intentionally teach students how to fail, persevere and respond with resilience (Lottero-Perdue & Parry, 2017). However, this goal cannot be achieved if the failure just appears at the end of the project when students do not have time to learn from their failure and improve their product. This explains the importance of getting students used to have a working product (even with a minimum set of functionalities) at any point in time during the development (Fronza et al., 2017).



Indeed, the undesirable output of an iteration can encourage tenacity and urge to improve. On the other hand, a tangible good result of the performed activities can increase self-esteem in students. Moreover, evidences about the students' progress support formative assessment. Finally, a plan-driven development process is not an option when the goal is to promote creativity, experimentation, and practical work in students. An incremental approach, instead, can help achieve these goals in the engineering field. The same goals are relevant to many disciplines in a teaching curriculum; therefore, the benefits of learning an Agile approach can be applied outside the context of SE.

Burnett and Myers (2014) proposed to frame the EUSE problem more along the lines of the problem-solving activity rather than the SE lifecycle, by supporting end-users as they work:

- end-users work opportunistically, incrementally, and rarely one lifecycle phase at a time (Burnett & Myers, 2014);
- end-users exploratory mix programming, testing, and debugging, mostly by trial-and-error (Burnett & Myers, 2014);
- end-users are capable of constructing software by direct manipulation (Costabile et al., 2008);
- end-users very much prefer collaborative activities (Costabile et al., 2008).

The modern teaching approaches suggest to encourage and leverage the above-mentioned work habits. Examples of these teaching approaches are: collaborative learning (Barkley, Cross, & Major, 2014), learn-by-doing (Moye, Dugger Jr., & Starkweather, 2014), use of tangibles (Price, Rogers, Scaife, Stanton, & Neale, 2003), and project-based learning (Krajcik & Blumenfeld, 2006). Moreover, as shown in Table 1, the values in the Agile Manifesto<sup>1</sup> can be mapped to the K-12 learning environment, considering the above-mentioned end-user's habits and modern teaching methods (Stewart, DeCusatis, Kidder, Massi, & Anne, 2009). In this context, the instructor can play the role of a customer (Steghöfer et al., 2016). Indeed, considering the instructor as a customer (and not as a source of continuous support) can help in fostering students' self-organization on their projects by reducing their dependence on the instructor's assistance (Kastl et al., 2016).

Nevertheless, the waterfall development model has been for a long time the development strategy taught in schools and universities (Kropp & Meier, 2013). Switching to Agile, in fact, would require switching to an environment in which the *process* by which the students arrive at the product is emphasized (Steghöfer et al., 2016). Teaching Agile principles has recently drawn researchers' attention, but papers and experience reports in which the authors discuss their experiences are mostly set at university level (Alégroth et al., 2015; Astrachan, Duvall, & Wallingford, 2001; Mahnic, 2012; Paasivaara, Heikkilä, Lassenius, & Toivola, 2014).

---

<sup>1</sup><https://www.agilealliance.org/agile101/the-agile-manifesto/>.

**Table 1** Mapping the agile manifesto to the K-12 environment

Values in the agile manifesto	K-12 environment
Individuals and interactions over processes and tools	End-users very much prefer collaborative activities. Student-centric learning environments should be favored, where students actively participate in activities and group-based components that reinforce concepts and allow for exploration (Stewart et al., 2009)
Working software over comprehensive documentation	An iterative environment leads to higher immersion in the project, more learning, and better-quality deliverables (Stewart et al., 2009)
Customer collaboration over contract negotiation	Greater access to the instructor can lead to more collaborative relationship (Stewart et al., 2009)
Responding to change over following a plan	End-users work opportunistically, incrementally, and rarely one lifecycle phase at a time (Burnett & Myers, 2014). They exploratory mix programming, testing, and debugging, mostly by trial-and-error (Burnett & Myers, 2014). Agility is the ability to adapt to different learning styles and change the delivery methods (Stewart et al., 2009)

### 3.2 Why Extreme Programming?

Agility at its core refers to a dynamic approach that removes all the a priori burdensome planning and design phases and substitutes them with an iterative approach that tackles smaller scale problems. Over the years, different agile methodologies have put into practice the Agile manifesto (Dingsyr, Nerur, Balijepally, & Moe, 2012), and each of them addresses specific needs and requirements. Extreme Programming (XP) is one of the methodologies that has integrated practices concerning the project management as well as the development process, by focusing on continuous communication and excellent programming habits (Beck, 2000). The main characteristics of XP are

- short cycles;
- incremental development;
- flexible implementation;
- high automation;
- evolutionary design;
- extensive communication;
- collaborative development.

By exploring the link between end-users and Agile (see Sect. 3.1), we found that XP can be an excellent candidate to teach software engineering to end-users in K-12 for two main reasons: (i) it does not require end-users to change their development habits radically, and (ii) it helps end-users to structure their development by introducing a lightweight set of practices.

When it comes to the application of XP, it is hard to classify teams if they are XP based only on whether they apply XP practices to the full or not. About this topic,

Kent Beck argues that “the full value of XP will not come until all the practices are in place. Many of the practices can be adopted piecemeal, but their effects will be multiplied when they are in place together” (Beck, 2000). The primary value of XP is indeed to establish a set of principles and practices to guide the development process; nevertheless, depending on the specific context, some teams (or individuals) may find that certain practices do not apply to them. It is important, however, to systematically recognize when each of the provided practices can be omitted and if there are any dependencies between them. Section 4 details how a set of XP practices can be leveraged or adapted to the K-12 educational context, and Sect. 5 explains when each practice (or group of practices) can be applied to achieve the goals set by XP.

### 4 Mapping XP Practices to K-12 Practices: A Toolbox

In the following, we describe a toolbox, which contains the most relevant XP practices in our context, and we show how they can be mapped to end-users’ working habits. Figure 1 shows how the working process is organized according to XP, and how each practice of the toolbox is mapped to the different phases of this iterative process.

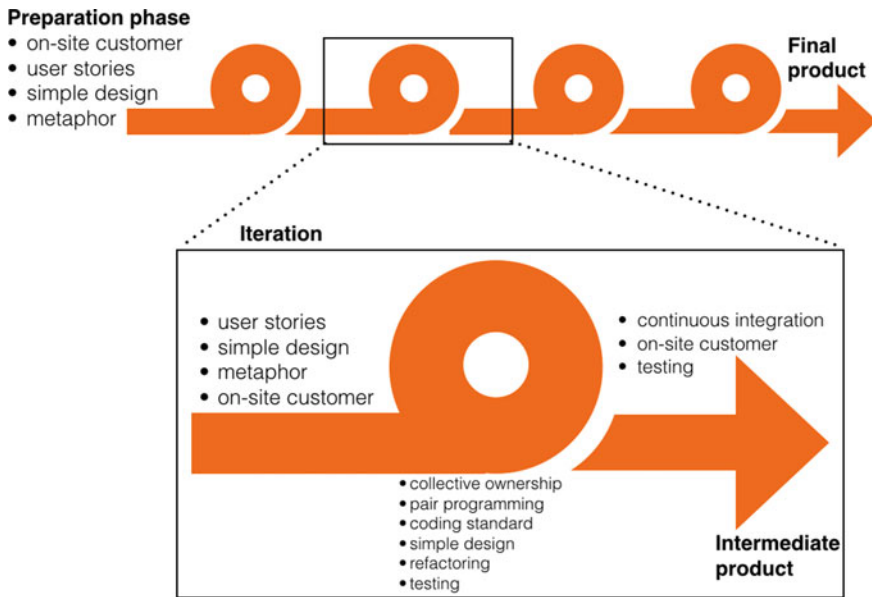


Fig. 1 Process description

## 4.1 User Stories

User stories represent the initial phase of the development process. The goal of user stories is to capture the requirements in a language that is understood both by developers and customers (Beck, 2000). They are written as a set of high-level scenarios that allow developers to understand the required features and the estimated effort. User stories also allow the classification of features, and the selection of those features that need to be implemented first and have a higher value to the customers. To this end, early, informal, paper prototypes are a very rapid way to generate ideas and obtain immediate feedback (Buxton, 2010; Traynor, 2012) to understand precisely how user stories will work and the client perspective. Moreover, paper prototypes are among the possible approaches (together with, for example, minimum viable product) to increase cost-effectiveness: it is indeed more cost-effective to make changes to a prototype than to an implemented user story (Fronza, El Ioini, & Corral, 2016c). For end-users, user stories can be mapped to the set of activities performed to understand and structure the problem domain, and to find an agreement on the requirements (Romeike & Göttel, 2012).

Images play an important role in human thinking, as they can capture visual and spatial information in a much more usable form than lengthy verbal descriptions (Thagard, 1996). In SE, rough, even hand-sketched, sequences of drawings (i.e., storyboards) are suggested to help understanding the customer's needs (Cardinal, 2013). In K-12 visuals are a common practice to support learners in: (i) linking new ideas to previous knowledge, (ii) connecting ideas, (iii) representing the structure of a product, and (iv) promoting collaboration (Walny, Carpendale, Riche, Venolia, & Fawcett, 2011). Therefore, user stories can be pretty easily applied in K-12, by simply leveraging tools already adopted in schools. The possible implementation of user stories in K-12 depends also on the type of system that is going to be created; for example, we illustrate storyboards, mind maps, execution trees, and role-playing.

### 4.1.1 Storyboard

A storyboard can be considered as the display of blocks of a comic strip, in which there is a visual representation of the sequence of an activity, which includes situations, actors, roles, and actions. In addition, a storyboard includes comments and annotations to provide a better notion of the action represented (van der Lelie, 2006). Regarding software development, storyboards can represent graphically a sequence diagram that uses the vocabulary provided by class diagrams, and it adds the chronological interaction between the different objects.

For example, when developing a mobile application in K-12, each panel of the storyboard can be considered as a *screen* of the app (Fig. 2a). Students are asked to draw each screen in a given format to create a mock-up prototype of the application's GUI. This requires defining the elements (i.e., figures, icons, text, background) of each screen and the actions that can be executed using those components (Fronza,

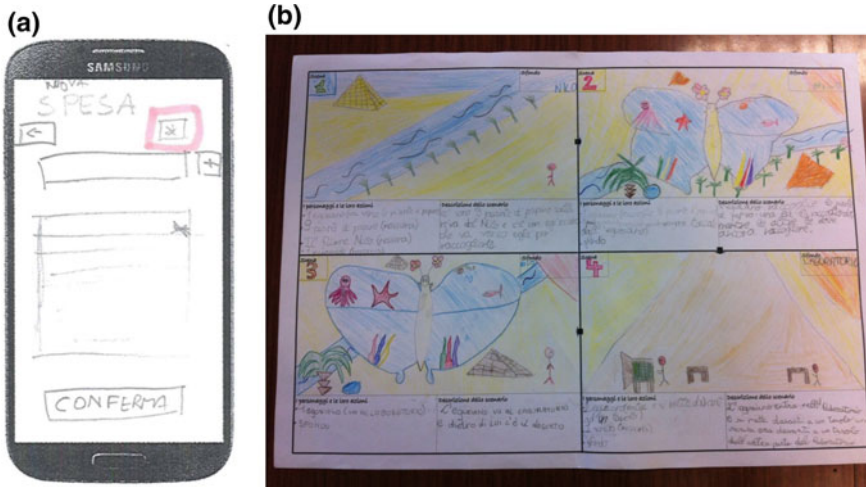


Fig. 2 Student-created storyboards in K-12

El Ioini, & Corral, 2015b, 2016a). Instead, when creating an animation, each panel of the storyboard represents a different scene (Fig. 2b).

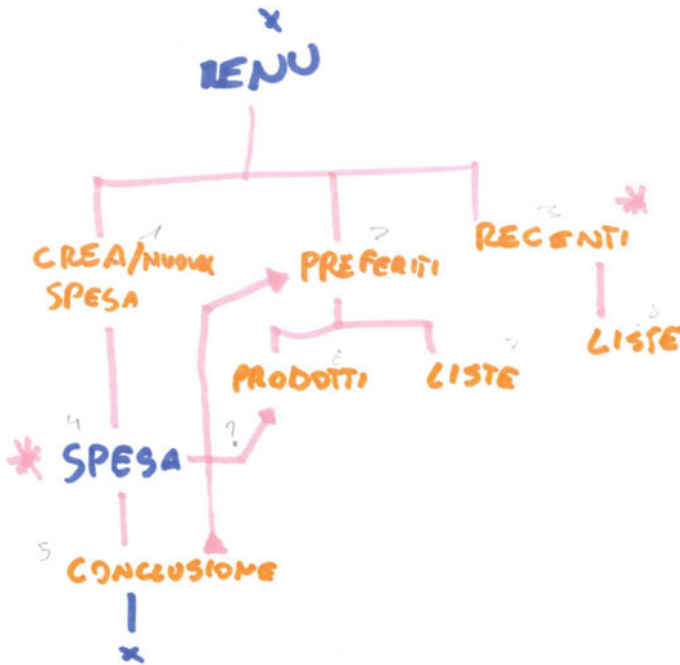
Positive results of the usage of storyboards in middle schools are reported in Fronza et al. (2017); in this case, a simple template is suggested (Fig. 2b), which includes: scene number, type of background, characters in the scene and their actions, short description. In the context of elementary schools, simpler stories are suggested (Fronza et al., 2016c), with clear scenes and fewer concepts. Storyboards can be used at the end of each iteration to check if the goal of the iteration has been achieved, by comparing the storyboard with the implemented scene.

#### 4.1.2 Execution Tree

The execution tree structures the execution flow in terms of a sequence. During the creation of a mobile app, for example, students can identify the transitions (i.e., edges of the tree) between the screens in the storyboard (i.e., nodes of the tree) (Fronza, El Ioini, & Corral, 2016b). To this end, students need to define what are the elements (e.g., buttons) and actions (e.g., tap) that trigger each transition. An example of the structure of an execution tree is shown in Fig. 3.

#### 4.1.3 Mind Map

Mind maps serve to organize ideas within a project, to identify their relevance, and to describe the relationships between them. One of the advantages of mind maps is that they are regularly used in K-12. Therefore, they can be used without specific training and without changing the end-users' working habits.



**Fig. 3** A student-created execution tree

Mind maps can be thought of as class diagrams that define the set of classes and the interactions among them. Starting from the requirements document, students can build the necessary vocabulary that will be used by all the subsequent activities. Moreover, drawing a mind map requires identifying the most important concepts (i.e., classes) in the requirements document, and also to group congruent concepts, find their properties (i.e., attributes) and draw connections among them. This fosters the creation of an end-user's mindset that would be open to Object-Oriented programming. The benefits of this activity in supporting students in the identification of the software elements and their interactions have been shown both in elementary (Fronza et al., 2016c) and in middle schools (Fronza et al., 2017).

#### 4.1.4 Role-Playing

Another possible implementation of user stories is role-playing, which supports requirements formulation from a user's perspective. In this activity, one student pretends to be the software and reacts accordingly, while another student takes the role of the user. The other students observe them in order to get details and constraints of the desired product. The process is repeated (with changing actors) until no more new requirements arise (Romeike & Göttel, 2012).

## 4.2 *Small Releases*

With the increasing demand on time to market products, software providers need to rapidly adapt to changes in order to maintain their competitive advantages. Small releases (a.k.a. short cycles) is the process of decomposing software development activities into short iterations allowing early and continuous feedback. For end-users, this is a fundamental requirement since it will enable them to work iteratively and keep adding more functionality to their core products. Additionally, in most cases, end-users use an exploratory approach in which they learn how to use the technologies and how to implement the required features. Thus, having short development cycles helps them evaluate their progress and adapt it accordingly. Moreover, this is particularly relevant for K-12, as it allows teachers to apply a formative assessment (Mikre, 2010).

A successful decomposition of the development process in small releases depends on the ability to estimate the required effort. This aspect might not be in the end-users' working habits. Therefore, they need to be supported in this process. One solution could be to link estimations to user stories with the disaggregation techniques (Cohn, 2005): the story's total estimate is calculated as the sum of the effort estimates of each task that is needed to complete it. Another possible approach is planning poker: Romeike and Göttel (2012) adopt professional card values with 15-min periods. Each student estimates the time to implement the user story in focus. Discussion of very divergent estimates is leveraged to clarify requirements and assumptions. After the planning poker is finished, the total estimated time is compared with the time available and user stories complexity is changed if needed.

What needs to be noted for these practices is that end-users might consider them as extra work that moves their attention from their goal. For this reason, it is important to let end-users perceive the usefulness of these practices, without forcing them too much to use them.

## 4.3 *Metaphors*

Metaphors as defined by Beck (2000), "a story that everyone customers, programmers, and managers can tell about how the system works", enable a shared vocabulary and common understanding on how the system functions. In the context of K-12, one of the early steps of the project can consist of establishing a shared vocabulary between teachers and the teams, especially when teams are not familiar enough with the application domain to be able to use native domain language.

## ***4.4 Coding Standards***

To increase productivity, developers are required to adhere to specific coding standards and conventions. This is particularly important for future modifications and refactoring. End-users tend to develop their own practices such as naming conventions and function compositions, however, in most cases they do not use any well-established reference when doing so. Using coding standards contributes to increased code quality (Fang, 2001), which is actually one of the EUSE goals. Therefore, end-users can benefit from learning this practice. For instance, teams can decide on naming conventions for variables and components. Another example concerns the testing process of the developed application: they can define a number of steps to follow and agree on adopting them.

## ***4.5 Collective Ownership***

In XP no one owns a specific part of the developed code, rather any developer can change any line of code to add functionality, fix bugs, improve designs or refactor (Beck, 2000). This allows for a constant code review and updates by all the members of the team. In K-12 students often prefer “repeatedly playing safe and choosing the role they feel most comfortable with, as opposed to stepping outside their comfort zone” (Stewart, 2014). Therefore, the risk is to have them working only on the piece of code they feel more comfortable with and then integrating the produced modules together. This might create situations in which no one sees the full picture of the project. Additionally, if team members are missing, it becomes difficult and time consuming to substitute them. Collective ownership in K-12 can be achieved by asking all team members to perform all the activities of the project; this means that no member is dedicated only to one task, rather they all need to be able to understand and modify the project tasks. Other XP practices (such as pair programming and coding standards) can support collective ownership.

## ***4.6 Simple Design***

The best design is the simplest that works. XP aims at finding the most straightforward solution that passes all the specified tests and meets the business values. One of the practices of XP is to write the tests first, which would fail initially, then write the minimum amount of code to make them pass (Beck, 2000).

K-12 teams tend to start by developing simple solutions for specific problems. However, when the scale of the problem increases, it becomes hard for them to manage. Two approaches are usually considered at this point: (i) overuse conditional statements to accommodate all the possible scenarios; (ii) develop different modules to tackle each scenario, then find a way to connect them to solve the bigger problem.



Good practices for simple design that K-12 can learn and apply are, for example, related to constructs, such as: making sure that the proper constructs are used (e.g., use loops instead of repeating a statement  $n$  times, use signals in Scratch<sup>2</sup> instead of timers), and use the minimum necessary constructs (classes, blocks, figures, etc.). To apply these practices, introductory exercises should be proposed with the goal of making sure that students learn the most relevant constructs for their activity (Fronza et al., 2017). Afterwards, code inspections at the end of each iteration can help in providing feedback to the students by, for example, suggesting the usage of a loop instead of repeated statements. Another good practice is to remove duplications from programs (e.g., use a function instead of implementing the same functionality in different parts of the code). In this case, high-level design (for example, using mind maps) can help students to identify those parts of the program that can be removed.

## 4.7 Refactoring

Refactoring is the process of improving code quality without changing its functionality (Beck, 2000). It consists of continuously reviewing the developed code and updating it (e.g., eliminate duplication, fix coding standards, naming conventions) manually or using dedicated tools. The main benefits of refactoring are to ease the maintainability and extendibility of the developed code. Refactoring can be triggered by the detection of code smells (Van Emden & Moonen, 2002) using automated tools, or it can take place at specific time intervals during the development process (e.g., end of each iteration). For K-12, refactoring is usually triggered by two factors: (i) adapting the existing code to support new features (e.g., changing functions parameters); (ii) substituting an existing construct with a new one (e.g., learning how to use messages instead of timers to move objects, setting the initial position of objects in a Scratch scene). Refactoring as considered in XP gives K-12 students a more comprehensive perspective and pushes them to plan for future extensions and higher maintainability. For instance, while building the different modules, functional decomposition can help them to increase the maintainability of the produced code; at the same time, in terms of extendibility, any future changes can focus on single modules rather than having to touch different parts of the code.

## 4.8 Testing

Testing is the process of executing the system functionality with the intent of finding bugs and errors (Myers, Sandler, & Badgett, 2011). In a software system, every part of the produced code needs to be thoroughly tested before it is released. Depending on the development process, different types of testing can be performed (e.g.,

---

<sup>2</sup><https://scratch.mit.edu>.

unit testing, integration testing, system testing). XP adds a new flavor to testing by introducing the concept of test first, which advocates writing tests as early as possible in the development lifecycle (Beck, 2000). This helps to have a well-structured testing framework and minimizes the testing effort in the long run. Acceptance tests is another important concept that allows developers to constantly having feedback from the customer. In K-12 context, teams tend to follow an opposite approach due to their inexperience, by continuously implementing and testing new functionality. In many cases, the nature of tools used by end-users forces them to follow this pattern. For instance, when using visual programming end-users need to implement and test each functionality to see if the added blocks work correctly (e.g., the right position, the right movement). User stories can be used as acceptance tests to validate the implemented functionality. For example, if each iteration's goal is the development of a panel in the storyboard, the comparison of the implemented software with the corresponding panel can indeed provide immediate evidence of the achievement of the iteration goal.

## 4.9 *Pair Programming*

In XP, code is developed in pairs: the *driver* writes the code, while the *navigator* reviews the code, thinks of the overall architecture, finds better solutions, and brainstorms (Beck, 2000). The central assumption is that having two developers work together will produce higher quality code, which reduces testing and debugging costs. Moreover, it has been shown that pair programming can improve software development under other perspectives, such as improving design, enhancing team communication, increasing job satisfaction, facilitating the integration of newcomers, and reducing training costs (Di Bella et al., 2013; Fronza & Succi, 2009).

In schools, it is very common to have two students sharing one computer due to limited hardware. This practice supports this need while adding a framework that encourages attention from both students, mutual learning, and a notion of programming as a social activity (Romeike & Göttel, 2012).

Pair programming can be used easily in K-12 as students prefer collaborative activities (Costabile et al., 2008). Nevertheless, some issues typical of the professional environment also apply in K-12. Therefore, teachers should carefully pick pairs by following some principles: for example, a very expert student in the pair might result in excluding the “weaker” student from the coding activities (Williams & Kessler, 2002).

## 4.10 *Continuous Integration*

The goal of continuous integration is to have at all times a working product. Once the functionality is implemented and tested, it is integrated with the core system. As we have seen in Sect. 3.1, in general, end-users work incrementally (Burnett & Myers,

2014). Asking K-12 teams to develop independent modules to integrate them at the end would not leverage this work habit. Moreover, in K-12 sometimes the nature of the used tools makes continuous integration the only possible solution; for instance, when using visual programming, users are forced to integrate the new features on top of the existing ones. When looking at XP practices, K-12 teams take advantage of the process of continuous integration by making sure that the new functionalities do not negatively affect or break the existing ones (Beck, 2000).

#### **4.11 On-site Customer**

For XP the customer is always present, in other words, the customer takes an active role in the development process (Beck, 2000). This allows quick and face-to-face feedbacks, which are at the heart of Agile methods. From user stories to acceptance tests, the customer helps developers in clarifying any doubt and assigns priorities to the different functionalities. In K-12, teachers can assume the customer role in order to guide the project progress. This helps the students to have a reference point in case of doubts or confusions. For instance, in line with the idea of the formative assessment, the teacher can help them when deciding the order of functionalities to implement or to validate an implemented feature.

### **5 Getting the Right Practice from the Toolbox: A Selection Strategy**

In the previous section, we have described a *toolbox of XP practices* that K-12 teachers can use in their classrooms. The goal of this section is to support teachers in selecting the right practices from this toolbox, based on different criteria.

For example, a teacher can select “collective ownership of code” so that all the members of the team will have a full picture of the project. However, taking such practice in isolation could lead to chaos, because anybody can change the system without constraints (Baird, 2002). To address this issue, XP suggests *using practices in a concerted way*, so that the weakness of a particular practice is mitigated by the other balancing practices. In our example, pair programming, coding standards, and testing should all be selected by the teacher to balance collective ownership. These practices are part of the so-called *programming* area, which describes how actual coding is done. Another two areas are defined, with some overlapping practices between the three areas. The area of *process* describes how the development activities are organized, and includes on-site customer, testing, and small releases. The third area deals with *team* management and includes collective ownership, continuous integration, metaphor and coding standards, pair programming, and small releases.

However, XP is not just a mechanical assembling of practices. Instead, it is built on values and principles. Indeed, XP practices can be grouped together, reflecting how they relate back the principles of XP. Table 2 lists the XP practices and relates them to the underlying core principles (Baird, 2002). For example, testing, on-site customer, and pair programming all relate to fine scale feedback. These XP principles can find an application in the K-12 environment to apply formative assessment; in this case, teacher and learners need rapid feedback to modify teaching and learning activities based on the performed assessment.

To provide a concrete guide to choose the right set of practices to apply in a given project, we propose a *selection strategy* that uses the project goal and principles to support the decision making.

During the selection process, teachers need first (Stage 1 in Table 2) to choose the goal of the project, being one of the three above-mentioned areas: programming, process, or team. The project goal expresses what aspect of the project we are going to focus on. For instance, developing a project with programming as its target indicates that the goal is to make students learn programming practices, while a project with the team as its goal focuses on teaching students how to work as a team independently from the task at hand. Choosing the project goal activates a set of practices that can be used for each specific goal. The second selection criterion is the “principles to foster” (Stage 2 in Table 2). Each of the XP principles is mapped to a set of practices, and by choosing one or more principles, a new set of practices is activated. We note that teachers can also combine more options from each category (e.g., project goal: programming and process).

Once the selection is done, teachers need only to consider the set of activated practices that reflect the team type; in other words, if some of the activated practices can only be used by teams (i.e., those practices indicated in Table 2 with \*), and the project will be undertaken individually, those practices need to be omitted.

*Example 1* The teacher defines the project requirements as follows: The project goal is to learn the *process* of analyzing data coming from a set of sensors. During the project development, students will be working *individually*. The teacher wants to provide *fine scale feedbacks* and have a *shared understanding* between the students and the teacher (i.e., the customer) of all the project components. Table 3 shows the practices selected from the toolbox: in stage 1 and 2 all the practices corresponding to the project goal and principles are activated, but since the projects will be developed individually, the practices that can only be done in teams are omitted (i.e., those represented in Table 2 with \*).

## 6 Conclusion

End-user programming has empowered millions of end-users to create their own software. End-User Software Engineering (EUSE) aims to increase the quality of end-user-created software by extending the benefits of a SE approach beyond the

**Table 2** A selection strategy for practices

		Practices in the toolbox									
		1	2	3	4	5	6	7	8	9	10
Stage 1	Goals			✓		✓	✓	✓			
	Programming	✓	✓								
Stage 2	Process		✓					✓			
	Team		✓	✓	*				*	✓	
	Fine scale feedback	✓	✓					✓	*	✓	✓
	Continuous process rather than batch		✓				✓			✓	
Shared understanding				✓	*						
1. User stories		6. Refactoring									
2. Small releases		7. Testing									
3. Metaphor and coding standard		8. Pair Programming									
4. Collective ownership		9. Continuous integration									
5. Simple design		10. On-site custome									

*Legend* ✓: practices for both teams and solo; \*: practices only for teams

**Table 3** Selection strategy applied to Example 1

Stage 1	Goal	Process: learn the process of analyzing data coming from a set of sensors
Stage 2	Principles	Fine scale feedback Shared understanding
	Team type	Individual development
	Selected practices	User stories Small releases Metaphor and coding standard Simple design Testing Continuous integration On-site customer

professional field. This chapter motivates the choice of XP as a good candidate for end-users in K-12 among the possible Agile methodologies. Moreover, the chapter shows that limited examples of Agile training in K-12 exist; most of these studies apply a limited number of XP practices in isolation, thus ignoring the dependencies between them.

The contribution of this chapter is twofold. First, it provides a description of each practice and reports examples and guidelines from existing studies. Second, the chapter proposes a XP-toolkit to encourage a synergistic use of XP practices that respects the dependencies between them. As an additional contribution, the chapter contributes to the End-User Software Engineering research field by encouraging further case studies with sets of practices, which could provide additional evidences on the positive effects of the toolkit in particular, and of bringing Agile inside the classroom in general.

## References

- Alégroth, E., Burden, H., Ericsson, M., Hammouda, I., Knauss, E., & Steghöfer, J.-P. (2015). Teaching scrum—What we did, what we will do and what impedes us. *Proceedings of XP*, 212, 361.
- Astrachan, O., Duvall, R. C., & Wallingford, E. (2001). Bringing extreme programming to the classroom. In *XP Universe* (Vol. 2001).
- Baird, S. (2002). *Extreme programming practices in action*. Pearson Education, Informit. Retrieved from <http://www.informit.com/articles/article.aspx?p=30187>.
- Barkley, E. F., Cross, K. P., & Major, C. H. (2014). *Collaborative learning techniques: A handbook for college faculty*. Wiley.
- Beck, K. (2000). *Extreme programming explained: Embrace change*. Addison-Wesley Professional.
- Burnett, M. (2009). What is end-user software engineering and why does it matter? In *International Symposium on End User Development* (pp. 15–28).
- Burnett, M. M., & Myers, B. A. (2014). Future of end-user software engineering: Beyond the silos. In *Proceedings of the on Future of Software Engineering* (pp. 201–211).

- Buxton, B. (2010). *Sketching user experiences: Getting the design right and the right design*. Morgan Kaufmann.
- Cardinal, M. (2013). *Executable specifications with scrum: A practical guide to agile requirements discovery*. Addison-Wesley.
- Chimalakonda, S., & Nori, K. V. (2013). What makes it hard to teach software engineering to end users? Some directions from adaptive and personalized learning. In *2013 IEEE 26th Conference on Software Engineering Education and Training (CSEE&T)* (pp. 324–328).
- Cohn, M. (2005). *Agile estimating and planning*. Pearson Education.
- Costabile, M. F., Mussio, P., Parasiliti Provenza, L., & Piccinno, A. (2008). End users as unwitting software developers. In *Proceedings of the 4th International Workshop on End-User Software Engineering* (pp. 6–10). New York, NY, USA: ACM.
- Di Bella, E., Fronza, I., Phaphoom, N., Sillitti, A., Succi, G., & Vlasenko, J. (2013). Pair programming and software defects—A large, industrial case study. *IEEE Transactions on Software Engineering*, 39(7), 930–953.
- Dingsyr, T., Nerur, S., Balijepally, V., & Moe, N. B. (2012). A decade of agile methodologies: Towards explaining agile software development. *Journal of Systems and Software*, 85(6), 1213–1221 (Special Issue: Agile Development).
- Fang, X. (2001). Using a coding standard to improve program quality. In *Proceedings of the Second Asia-Pacific Conference on Quality Software, 2001* (pp. 73–78).
- Fronza, I., El Ioini, N., Janes, A., Sillitti, A., Succi, G., & Corral, L. (2014). If I had to vote on this laboratory, I would give nine: Introduction on computational thinking in the lower secondary school: Results of the experience. *Mondo Digitale*, 13(51), 757–765.
- Fronza, I., El Ioini, N., & Corral, L. (2015a). Students want to create apps: Leveraging computational thinking to teach mobile software development. In *Proceedings of the 16th Annual Conference on Information Technology Education* (pp. 21–26).
- Fronza, I., El Ioini, N., & Corral, L. (2015b). Students want to create apps: Leveraging computational thinking to teach mobile software development. In *Proceedings of the 16th Annual Conference on Information Technology Education* (pp. 21–26). New York, NY, USA: ACM.
- Fronza, I., El Ioini, N., & Corral, L. (2016a). Blending mobile programming and liberal education in a social-economic high school. In *International Conference on Mobile Software Engineering and Systems, MOBILEsoft 2016* (pp. 123–126).
- Fronza, I., El Ioini, N., & Corral, L. (2016b). Computational thinking through mobile programming. In *International Conference on Mobile Web and Information Systems* (pp. 67–80).
- Fronza, I., El Ioini, N., & Corral, L. (2016c). Teaching software design engineering across the K-12 curriculum: Using visual thinking and computational thinking. In *Proceedings of the 17th Annual Conference on Information Technology Education* (pp. 97–101).
- Fronza, I., Ioini, N. E., & Corral, L. (2017). Teaching computational thinking using agile software engineering methods: A framework for middle schools. *ACM Transactions on Computing Education (TOCE)*, 17(4), 19.
- Fronza, I., & Succi, G. (2009). Modeling spontaneous pair programming when new developers join a team. In *Proceedings of the 10th International Conference on Agile Processes and Extreme Programming in Software Engineering (XP2009), Pula, Italy, May 2009*.
- Gross, P. A., Herstand, M. S., Hodges, J. W., & Kelleher, C. L. (2010). A code reuse interface for non-programmer middle school students. In *Proceedings of the 15th International Conference on Intelligent User Interfaces* (pp. 219–228).
- Kastl, P., Kiesmüller, U., & Romeike, R. (2016). Starting out with projects: Experiences with agile software development in high schools. In *Proceedings of the 11th Workshop in Primary and Secondary Computing Education* (pp. 60–65). New York, NY, USA: ACM.
- Krajcik, J. S., & Blumenfeld, P. C. (2006). *Project-based learning*.
- Kropp, M., & Meier, A. (2013). Teaching agile software development at university level: Values, management, and craftsmanship. In *2013 IEEE 26th Conference on Software Engineering Education and Training (CSEE&T)* (pp. 179–188).

- Lottero-Perdue, P. S., & Parry, E. A. (2017). Perspectives on failure in the classroom by elementary teachers new to teaching engineering. *Journal of Pre-College Engineering Education Research (J-PEER)*, 7(1), 4.
- Mahnic, V. (2012). A capstone course on agile software development using scrum. *IEEE Transactions on Education*, 55(1), 99–106.
- Meerbaum-Salant, O., & Hazzan, O. (2010). An agile constructionist mentoring methodology for software projects in the high school. *ACM Transactions on Computing Education*, 9(4), n4.
- Mikre, F. (2010). The roles of assessment in curriculum practice and enhancement of learning. *Ethiopian Journal of Education and Sciences*, 5(2).
- Monteiro, I. T., de Castro Salgado, L. C., Mota, M. P., Sampaio, A. L., & de Souza, C. S. (2017). Signifying software engineering to computational thinking learners with AgentSheets and polifacets. *Journal of Visual Languages & Computing*, 40, 91–112.
- Moye, J. J., Dugger, W. E., Jr., & Starkweather, K. N. (2014). “Learn by doing” research: Introduction. *Technology and Engineering Teacher*, 74(1), 24–27.
- Myers, G. J., Sandler, C., & Badgett, T. (2011). *The art of software testing*. Wiley.
- Paasivaara, M., Heikkilä, V., Lassenius, C., & Toivola, T. (2014). Teaching students scrum using lego blocks. In *Companion Proceedings of the 36th International Conference on Software Engineering* (pp. 382–391). New York, NY, USA: ACM.
- Price, S., Rogers, Y., Scaife, M., Stanton, D., & Neale, H. (2003). Using tangibles to promote novel forms of playful learning. *Interacting with Computers*, 15(2), 169–185.
- Romeike, R., & Göttel, T. (2012). Agile projects in high school computing education: Emphasizing a learners’ perspective. In *Proceedings of the 7th Workshop in Primary and Secondary Computing Education* (pp. 48–57). New York, NY, USA: ACM.
- Steghöfer, J.-P., Knauss, E., Alégroth, E., Hammouda, I., Burden, H., & Ericsson, M. (2016). Teaching agile: Addressing the conflict between project delivery and application of agile methods. In *Proceedings of the 38th International Conference on Software Engineering Companion* (pp. 303–312).
- Stewart, G. (2014). *Promoting and managing effective collaborative group work*. Belfast Education and Library Board. Retrieved from <http://www.belb.org.uk/Downloads/iepdpromotingandmanagingcollaborativegroupworkmay14.pdf>.
- Stewart, J. C., DeCusatis, C. S., Kidder, K., Massi, J. R., & Anne, K. M. (2009). Evaluating agile principles in active and cooperative learning. In *Proceedings of Student-Faculty Research Day, CSIS, Pace University*.
- Thagard, P. (1996). Cognitive science.
- Traynor, B. (2012). Rapid paper prototyping: 100 design sketches in 10 minutes, 18 designs presented, 6 prototypes tested, student engagement-priceless! In *2012 IEEE International Conference on Professional Communication Conference (IPCC)* (pp. 1–5).
- Umarji, M., Pohl, M., Seaman, C., Koru, A. G., & Liu, H. (2008). Teaching software engineering to end-users. In *Proceedings of the 4th International Workshop on End-User Software Engineering* (pp. 40–42).
- van der Lelie, C. (2006, April 01). The value of storyboards in the product design process. *Personal and Ubiquitous Computing*, 10(2), 159–162.
- Van Emden, E., & Moonen, L. (2002). Java quality assurance by detecting code smells. In *Ninth Working Conference on Reverse Engineering, 2002. Proceedings* (pp. 97–106).
- Walny, J., Carpendale, S., Riche, N. H., Venolia, G., & Fawcett, P. (2011). Visual thinking in action: Visualizations as used on whiteboards. *IEEE Transactions on Visualization and Computer Graphics*, 17(12), 2508–2517.
- Williams, L., & Kessler, R. (2002). *Pair programming illuminated*. Addison-Wesley.
- Ye, Y., & Fischer, G. (2007). Designing for participation in socio-technical software systems. *Universal Access in Human Computer Interaction. Coping with Diversity* (pp. 312–321). Longman Publishing Co., Inc.



**Part III**  
**Reconceptualising Learning Environments**  
**Using Agile and Lean Approaches**

# Lean and Agile Higher Education: Death to Grades, Courses, and Degree Programs?



Guttorm Sindre

**Abstract** Most universities provide education in a traditional plan-based manner, for instance with rigid degree programs that force students to make big decisions up front. Lean and agile education would rather operate in small increments and allow the students to make many iterative decisions along the way. This chapter discusses how lean and agile education might be radically different from plan-based education, what obstacles there are to lean and agile education, and how information technology could reduce these obstacles. Ultimately, information technology supporting a fine-granular matching of student learning outcomes with competencies needed by employers could enable agile study choices by the students themselves that would be superior even to the most thoroughly planned degree program, because the quick pace of technology progress means that it will be almost impossible to predict future work–life needs in detail.

**Keywords** Lean education · Agile education · Grading · Courses  
Degree programs · Employability · Flexibility

## 1 Introduction

Consider a student a year or two into a degree program, who wants another path of study. There could be a number of different reasons. Maybe the chosen discipline seemed like a good idea at enrollment, but the job market turned? Maybe a worsened family economy means that the student needs to start working sooner? Maybe the student has realized that the discipline or pedagogical approach of the degree program was a poor match with personal talents and preferences. Maybe health problems or otherwise limited learning capacity means that the student needs longer time, a different pace than the majority of the class? Unfortunately, most such changes are

---

G. Sindre (✉)

Department of Computer Science, Norwegian University of Science  
and Technology (NTNU), Trondheim, Norway  
e-mail: [guttorm.sindre@ntnu.no](mailto:guttorm.sindre@ntnu.no)

© Springer Nature Singapore Pte Ltd. 2019

D. Parsons and K. MacCallum (eds.), *Agile and Lean Concepts for Teaching  
and Learning*, [https://doi.org/10.1007/978-981-13-2751-3\\_8](https://doi.org/10.1007/978-981-13-2751-3_8)

155

difficult to accommodate in the currently dominant system of rigid degree programs. Changing to another degree program may be possible, but often with substantial loss of time. Leaving university early means leaving without a degree, which really hurts employability (Nguyen, 2012). On the other hand, many young people have found that a university degree was no guarantee for a job either (Coppola & O'Higgins, 2017), and for many of those who do get jobs there may be a substantial mismatch between the education and work–life needs (Caroleo & Pastore, 2017). It has been argued that education needs to become more flexible to reduce mismatch and dropouts (Moreno Minguez, 2013). One way to become more flexible could be to take a lean and agile approach to education.

Although educational discourse is currently dominated by an employability perspective (Nilsson & Nyström, 2013), education may also be pursued primarily for personal development, according to the Humboldtian idea of *Bildung* rather than *Ausbildung*. According to Biesta (2009), education can serve three different purposes, *qualification* (for jobs, or more generally as a citizen), *socialization*, and *subjectification*. Even for the latter two purposes, it can be argued that globalization and high-tech society are causing an acceleration of social changes (Rosa, 2003), making it more challenging than before to know what kind of education might best meet these purposes some years ahead. Whenever it is difficult to know the requirements up front, there is a possible case for lean and agile.

Lean and agile education are not new ideas, as there have been proposals in that direction at least since the turn of the millennium (Alp, 2001; Sokolov, 2001), about the same time as the well-known manifesto for agile software development (Beck et al., 2001) was published. Whereas agile software development has enjoyed high and increasing popularity (Dybå & Dingsøy, 2009), as has lean software development (Poppendieck & Cusumano, 2012), lean and agile has had much less impact in education so far. True, there have been proposals for lean and agile both for course delivery (Chun, 2004; D'Souza & Rodrigues, 2015; Emiliani, 2004; Whalen, Freeman, & Jaeger, 2008) and degree program design (El-Abbassy, Muawad, & Gaber, 2010; Emiliani, 2005). However, the mainstream university practice remains a traditional plan-based one.

Many of the proposals for lean and agile higher education mentioned above have focused on how the university can become more lean and agile. This paper suggests that the *students* have the primary need to be lean and/or agile. It is they who will be facing a rapidly changing society (Rosa, 2003) and job market due to exponential technological progress (Brynjolfsson & McAfee, 2014; Frey & Osborne, 2017). They may thus have much bigger trouble than previous generations in determining up front what education will be best for them. Universities may only need to change to offering more lean and agile education if there is a sufficient demand from students requiring such education, or if they start being outcompeted by new actors in the education system.

This chapter asks, and tries to respond to, the following questions:

- What would be the key differences between lean and agile on the one hand, and traditional plan-based university education on the other hand? This question can be considered both on the level of courses, and on the level of whole degree programs.
- What are the key obstacles towards lean and agile in higher education, i.e., reasons that universities may want to continue with their current plan-based approach?
- How can IT enable a transition to lean and agile studies?
- What may this transition eventually mean for some aspects of education that many take for granted today: grades, courses, and degree programs?

The next four sections will discuss each of these questions, the last of them also drawing some concluding remarks.

## **2 Lean and Agile as a Departure from Plan-Based Education**

### ***2.1 Meaning of Lean and Agile***

Lean and agile share a preference for lightweight, iterative, and incremental development, as opposed to more plan-based approaches, which require more heavyweight documentation (Poppendieck & Cusumano, 2012). At the same time, lean and agile are not synonymous. Lean has a key focus on reducing waste, whereas agile emphasizes the ability to adapt quickly to changes (Hallgren & Olhager, 2009). Hence, in industry, lean tends to be associated with a cost-leadership strategy, agile with a differentiation strategy. Making the parallel to education, the common denominator of lean and agile education would be the iterative and incremental aspects, letting students pursue their education in an emergent manner, as opposed to the traditional plan-based education dominant today. Lean education in particular would strive to contain only what is necessary (i.e., focus on waste reduction and cost leadership), while agile education in particular would cater especially to students for whom cost might be less of an issue, but who might want to differentiate themselves from the pack and get an education adapted to their personal preferences.

What might the lean principle of waste reduction mean in education? The first and most obvious interpretation would be that the education is cheap in terms of small or zero tuition fees, as is typically the case for MOOCs. On the other hand, it might also mean that the education is quick. Even when tuition is free, education will have a cost for the student in the sense that the capacity for paid work is reduced while studying—a loss hopefully regained by higher earnings afterwards. The longer an education takes, the bigger the up-front investment, and the bigger the risk that the assumed return on investment somehow does not materialize. A strict adherence to lean might thus inspire attempts to strip the education of any content not strictly necessary for the purpose, thus making it quicker and cheaper.

**Table 1** Key differences between plan-based and agile/lean approaches

Plan-based education	Lean and agile education
Big (and often fixed size) courses	Smaller courses or creditable units
Rigid degree programs, forcing Big Decision Up Front by students	More flexible degrees, allowing for iterative decisions by students
Detailed specification of programs and courses up front	Limited specification, delaying decisions until the time of learning
Hard to find starting degrees of less than 3 years' duration	<b>Lean:</b> Also offering shorter degrees, 1 year or less
Primarily thinking full-time study, then full-time work	Encouraging students to interleave studies and work
"One size fits all" instruction	<b>Agile:</b> Offering instruction more personalized to the individual student

The agile approach might instead be to choose an educational path which holds the door open for many different types of jobs, allowing for quick changes of direction, either because the job market changes, or because the student's preferences and self-insight evolve. The crowning example of a differentiation strategy might be an education that is adapted to the preference of one single person. For instance, a new kind of job emerges due to some technological change, and a student perceives this as the dream job, thus seeking a personally adapted education for this.

## 2.2 Key Differences from Plan-Based Education

Traditional, plan-based university studies tend to inhibit lean and agile behavior by students, because most degree programs offer limited maneuverability. Table 1 provides an overview of some key differences between plan-based and lean/agile, to be explained in the next paragraph. Most of the rows are equally relevant both for lean and agile; those that are more specific have **Lean** or **Agile** in bold in the rightmost column.

Course sizes vary between countries or universities, but a typical size may be 200–300 person-hours of nominal effort per student per course, and a typical duration a whole trimester or semester, since most schools tend to run several courses in parallel rather than all in sequence. Although students may learn iteratively and incrementally within a course, the course will likely be the smallest unit for which it is possible to earn credit. Leaving a course midway will not yield half credit but zero credit. The increment size and cycle time is thus much bigger than in agile software development, where a sprint tends to last for 2–4 weeks. Thus, the smallest delivery unit in education tends to be quite bulky, reducing maneuverability for students, making it hard to move through one's studies in an agile manner.

Degrees have a similar problem on a larger scale: Leaving university with half a degree is almost like leaving with nothing at all, often detrimental for employability (Nguyen, 2012). Many young people fresh out of secondary education have limited insight in academic disciplines, the job market, and even of themselves, and thus struggle to make meaningful choices of higher education (Holmegaard, Ulriksen, & Madsen, 2014). Yet, the main offering being degree programs lasting for three years or more, we are forcing a “Big Decision Up Front” upon these young people. The consequences of a wrong choice may be costly—for the student, and for society, for instance in terms of high dropout rates (O’Keeffe, 2013) or mismatch in the transition from one degree program to another, which delays the candidate’s graduation.

Similarly, there is a “Big Design Up Front” by faculty responsible for the degree program. Most universities require that programs are specified in detail, in terms of learning outcomes, compulsory and elective courses, etc. In most universities, this must go through quite some bureaucracy before enrolling students. Hence, it takes maybe 5–7 years from starting the design of a degree program until the first candidates graduate. It is difficult even for experts in the field to predict what will be the optimal package of learning outcomes and courses. This problem may become worse as the pace of change in the job market increases (Brynjolfsson & McAfee, 2014; Frey & Osborne, 2017).

Especially students who prefer a lean approach—getting as quickly as possible to a good job—will struggle to find the ideal program in the current university system. In many universities, the first obtainable degree is the Bachelor, lasting 3–4 years. Lean students might prefer much quicker programs, maybe just one year, maybe even less. While many universities do offer some one-year packages, these tend to consist of general introduction courses as preparation for further studies—meant for students who have not decided what to study yet, rather than students who want to get quickly to a job. On the other hand, so-called *nanodegrees* (Lemoine & Richardson, 2015; Waters, 2015) have recently been introduced to quickly educate people to fill jobs where industry feels an urgent lack of supply from the education sector. Such educational offerings emerged as a collaboration between industry and MOOC providers rather than from the university sector.

### ***2.3 Pursuing the Minimum Viable Competence***

Extrapolating a development in the direction of smaller courses and more flexible degrees, one might ask: Why study full time for 3 years or more before using any of the competence in a job? In agile software development, one feature is developed and shown to the customer to get feedback, then another feature is developed, etc. Similarly, the lean or agile learner could spend a month in university developing some competence, then for the next month take a job where that competence is used in practice, then come back to university to learn some more, then another job for a little while. Such iterative cycles would give the student—and the university—useful feedback on several levels: Was the acquired competence ideal for the job, or should

it have been something else? Especially for the student: Was this a type of job I would like to continue doing, or should I pursue other types of competence to qualify for another set of jobs? Especially for young people who are somewhat tired of studying after high school, but still need to study something because it is getting increasingly hard to find a job just on the basis of a high school diploma, this might be a better solution than the current full-time study, full-time work situation. Interleaving study and work, the student would resemble a lean startup. Just like a lean startup seeks to identify the *minimum viable product* to launch on the market (Moogk, 2012), the student (with the help of the university) might seek to identify the *minimum viable competence* that could lead to an interesting job. Such an offer might also be attractive for more experienced people whose previous job has disappeared, and who need to reeducate themselves.

A consideration of minimum viable competence might also be relevant in more traditional degree programs where students do want to keep studying for three years or more. Experiential learning, e.g., via internships and summer jobs, can be important both for motivation and the student's holistic learning beyond and between the separate courses in a degree program (Cantor, 1997)—as long as the job is *relevant*. The ability to get a relevant job early on (e.g., already at the first summer break, rather than just the second or third) might depend on how the degree program is structured. Hence a potentially valid question for degree program designers: What is the minimum viable competence that our students would need to qualify for relevant summer jobs already after the first or second year? If the degree program just contains general theory and nothing hands-on at this stage, students might not be so likely to get those jobs.

### 3 Obstacles to Lean and Agile

So far, we have argued for advantages of lean and agile education, but there are also challenges, some important examples listed in Table 2.

One potential advantage of traditional plan-based education is economy of scale (Laband & Lentz, 2003). It will be much cheaper for a university to have many students take the same course and grade them by the same assessment, than to let students make individual choices about how much to take of the course and when to be assessed. Furthermore, it will be much cheaper—and easier for resource allocation, employment planning, etc.—to run large groups of students through rigid degree programs, all taking the same package of courses, than to have the same number of students select different configurations of courses, drifting in various directions within and across disciplines. With smaller units of credit, there may also be a need for more assessments and grades, thus increasing the workload. In many countries, university funding (whether by tuition fees or covered by the state/country) will tend to increase the more course credits the university produces. If so, it actually does not pay off for universities to get candidates more quickly to work. There is better income in keeping them at university for as long as possible before they move on.

**Table 2** Challenges of lean and agile education

Lean/agile property	Challenge
Agile: Allowing students more flexibility and variability	For university: Poorer economy of scale, increased teaching workload, difficult resource planning For students: Class erodes
Smaller creditable units	More assessments, more grading
Students making their own choices	Risk of incoherent choices, e.g. prioritizing fun and easiness rather than relevance
Lightweight design of courses and degree programs up front	Limited QA, selection of topics driven by buzzwords?
Lean: Getting students quickly to jobs	Reduced university income with current funding models?

From a student perspective, a possible advantage of a rigid degree program is that it places every student in a class. This can have a social function, making friends with other students who take largely the same courses. While a fixed package of courses may sometimes feel like a straitjacket, it can also function as a guarantee. Enrolled in the degree program, you know that there is place for you on these courses. With free choices comes the increased risk, perhaps, that some courses you want to take are overbooked, if they have limited capacity for instance due to lab work with limited places. Moreover, freedom to choose only benefits the students if they have sufficient information and knowledge to make good choices. Otherwise it may be stressful to have to select your next course all the time, and there is a risk that some might end up with poor choices and thus a less useful competence profile than what they would have had with a rigid, faculty designed degree program.

The idea of just having a lightweight or rudimentary design of a degree program up front and then delay the rest of decisions until later, when more is known about work–life needs, may sound like a good idea in theory. However, it demands that stakeholders are capable of running this iterative process in a satisfactory way. Hence, faculty, students, and industry representatives would need an ongoing dialogue to make decisions for each next semester. What if the industry representatives are too busy, so it is hard to get good input—and faculty are also too busy, for instance with research, funding applications, etc.? This could end up with hurried decisions and exaggerated focus on the latest buzzwords in the discipline, while the knowledge profile of the candidates ends up less suitable than it would have been with more thorough planning up front.

Next, we will discuss how information technology might help address these challenges.



## 4 IT as an Enabler for Lean and Agile Education

At the outset of this section it is important to notice that we agree with those who emphasize that increasing usage of IT in education is not a goal in itself (Salomon, 2016). Rather, there must be some educational rationale, and an idea about how IT can support this rationale. In this chapter, the rationale is that education may need to become more flexible, due to the increased pace of change in technology and society—and this increased flexibility will be very costly unless supported by technology. Table 3 shows some identified challenges with lean and agile education, along with indications of how information technology can reduce the mentioned problem.

### 4.1 *E-learning and Crowdsourcing*

“One-size fits all” (e.g., one lecture series, one set of compulsory coursework, final exam with the same questions for everybody) is a well-known way to run a course, and offers economy of scale if there are many students taking the course. Allowing variation, such as students going at different paces, selecting different subsets of the learning goals of the course, craving different learning methods and resources, will be burdensome for the teacher—if it is still the same teacher burdened with the whole offering. However, why should the teacher take on this challenge alone? Except for some very specialized subjects (which tend to be taught in small classes, thus offering limited economy of scale anyway), and subjects inherently local to one particular country (such as, say, Norwegian law), most university courses have more or less overlapping siblings in other universities, globally amounting to hundreds or thousands. It should be unnecessary for thousands of teachers to spend lots of time

**Table 3** Mitigation of challenges by IT

Challenge	IT enabled solution
Agile: Increased workload for teachers if offering diversity	E-learning and crowdsourcing
Increasing number of tests and grades with smaller courses	E-exams, automated grading, peer review
Students struggling to make good choices	Ontologies of learning outcomes and job competence needs Job marketplace Learning analytics
Teachers struggling to make good choices, poor QA	Same as above
Lean: Funding models not promoting quick studies	New funding models?

developing the full set of learning resources for such a course. Instead, a big group of teachers with overlapping topics could make a joint effort to develop a comprehensive set of learning resources: traditional reading materials, interactive reading materials, brief screencast videos, longer lectures, self-test quizzes, problem-solving exercises and project tasks with suggested solutions, exam questions of various genres, etc. With a huge pool of shared resources, each teacher can recommend those that fit best for the local students, or the students may find resources themselves, for instance preferring a number of short screencasts combined with self-test quizzes to the offered two hour monologue lectures. For teachers able to utilize the pool of available resources, as well as contributing to it and thus shaping it, the future may allow for more diverse learning opportunities for that teacher's students, and yet the teacher may not have to work harder than today developing and providing learning resources and activities.

Formative feedback to students may also be more complicated if students pursue diverse learning goals within a course. However, modern e-learning tools can provide automated support for the feedback to students (Denton, Madden, Roberts, & Rowe, 2008) and guide them towards learning resources that work for them based on their results so far (Baker & Inventado, 2014; Beetham & Sharpe, 2013; Ellis, 2017).

Similar arguments can apply to summative assessment (i.e., which is graded or given credit). With shorter iteration cycles and more fine-granular tests, one might need more test items—but the teacher does not have to write them all alone. Rather, teachers of the same subject globally could collaborate to make thousands of test items of various question type genres, covering various parts of the subject, at various levels of difficulty. Then, the individual teacher need only pick questions for each test, or in some cases not even pick but just specify the topical scope of the test and the wanted level(s) of difficulty, and then have questions drawn randomly from the question base for each student during the exam. To mitigate cheating, it is anyway better to draw questions randomly from a huge question base than to offer the same questions for everybody. This is especially true now that collaboration among candidates or with outsiders is not only achieved by peeping at the neighbor's answers or exchanging information via the toilet, but also via modern electronic equipment, for instance tiny wireless earpieces which are very hard to detect (Srikanth & Asmatulu, 2014). To develop such a big question base is beyond the individual teacher, so national or global collaboration about item development would certainly be the way to go for the typical invigilated school exam. Even with global collaboration on test items, grading remains a challenge. However, automated assessment technology has moved well beyond multiple choice and short answer questions that have long been automatable. Much more complex answers such as engineering designs (Nutter, Pavlidis, & Pepper, 2014), software programs (Hakulinen & Malmi, 2014), and various types of essays (Balfour, 2013) are now possible. With the rapid progress of AI, an increasing subset of learning outcomes may be tested and graded automatically—with the same or better reliability than humans could offer. Then, agile variation and diversity might be offered to students without drowning the teacher in assessment work.

## 4.2 *The Future of Grades and Grade Transcripts*

Another question, however, is if we *need* to keep on grading—whether it be done by humans or machines (Kohn, 1994). If learning and assessment cycles become more fine-grained, per learning outcome rather than bigger courses, it might suffice to do a pass/fail for each learning outcome. The difference between a stronger and weaker student would then not be the grades, but the stronger student achieving more—and higher level—learning outcomes, where the weaker student achieves fewer or just lower level outcomes. Ultimately, this might mean that the traditional grade transcript disappears, to be replaced by a much longer list of precise learning outcomes achieved by a candidate. Here, the reader might naturally ask: Why would this be an advantage? What employer would prefer to read a very long list of precise learning outcomes, instead of a 1–2 page grade transcript with course titles and letters A, B, C...? Less is more, is it not? For several reasons, maybe it is not:

First, the grade transcript is a dated artefact from the age when such documents had to be printed on paper and read manually by people. What employer would want to read transcripts manually nowadays? Few, especially if they have hundreds of applicants to evaluate. They would like to analyze them automatically, comparing candidates back to back via a software application to make a shortlist—and spend manual evaluation time only with people on the shortlist. While “less is more” could be true for the information shown to the human decision-maker on screen, “more is more” would apply to the underlying data on which the analysis is based. A traditional grade transcript often contains too little information for any sophisticated analysis. Grading systems differ from country to country, and grade levels differ among universities or teachers, some being kind, others strict. Course titles are sometimes only vague indicators of the content. Two different universities may have courses with the same name, but huge differences in content or level of ambition—or courses with different names but essentially the same content. Often, the course title may obfuscate some of the content. For instance, a Computer Science graduate from University X may have only one course in the transcript with the word “programming” in the title. However, there may also be substantial amounts of programming in the Algorithms course, in the User Interfaces course (e.g., front-end programming), in the Database course (e.g., back-end programming with embedded SQL), in the Software Engineering course, and in several other courses. At the same time, in University Y, courses with the same titles may have no programming at all, being more theoretical. If the graduate from Y has two courses named “programming”, this seems to be more than X had, but is really less, a deceptive impression from the grade transcripts. Hence, back-to-back comparison of candidates based on grade transcripts has the risk of being highly misleading, unless they went to the same university during the same period of time.

Furthermore, the grade transcript is a kind of static report where the analysis has been locked to one particular dimension, namely the course module—long after it has been established in industry that decision makers will prefer the freedom to analyze data according to multiple dimensions (Chaudhuri & Dayal, 1997). Assume

that a company wants to hire a programmer, thus being particularly interested in the applicants' programming skills. As argued above, programming might be "hidden" under several course titles. With a long and precise list of learning outcomes, on the other hand, the back-to-back comparison of hundreds of candidates according to knowledge and skills in programming would be feasible. The same would apply to other knowledge and skills that typically cut across many courses, such as writing skills and collaboration skills.

### ***4.3 Ontologies of Learning Outcomes***

Of course, even with long lists of precise learning outcomes, there might be challenges that various universities formulate these differently. To really facilitate knowledge sharing and back-to-back comparison of learning outcomes across universities, it would therefore be helpful to have an internationally agreed ontology (Gruber, 1995) of possible learning outcomes in various subject areas. This leads on to ideas similar to those expressed by LinkedIn CEO Jeff Weiner as quoted in (Clayton, 2014): "We want to have a profile for every member of the global work force [...]. We want to have a profile for every company in the world [...] and digital representation of every job in the world [...] of every skill required to obtain those jobs, a digital presence for every university in the world and we want to make it easy for every individual company and university to share their professionally relevant knowledge."

Thus, an ontology might not only help employers evaluating and comparing candidates when hiring, but also help students. In what (Craig & Williams, 2015) call the "competency marketplace", students might be able to see what jobs fit their competency profiles—and if they do not qualify for any jobs yet: What competencies are they lacking? Thus, the lean student who needs to get work quickly, could look for the shortest viable path to a job. The agile student may pursue competencies that are needed in many different jobs, to have a wider selection and be less vulnerable to change. Professors may also be informed by such a marketplace. If they find that the competencies they are teaching are suffering from a steep decline in interest from the job market, this may spur a willingness to change the content of courses and degree programs.

If such an ambitious marketplace becomes a reality, enabling the possible matching of jobs and students, it is unlikely that students would make incoherent choices of subjects. After all, most students want their education to qualify them for attractive jobs, so if free choice has been associated with bad choice in the past, it has likely been due to lacking information about the consequences of choices. With the help of modern IT, especially big data and artificial intelligence, this would become much clearer: What kinds of jobs am I becoming more qualified for by learning X? Are they jobs I might want? What kinds of jobs do I de-select by not taking subject Y? If suddenly the set of possible jobs shrinks a lot just by that one decision, the student may think again. It would likely also help motivation if you made an agile choice to learn something because you see it is necessary to qualify for some range of jobs,

rather than just because it is included in some compulsory course in a rigid, traditional degree program. However, such a global candidate and job marketplace might entail severe privacy concerns, giving detailed accounts of what candidates have accomplished during their studies, possibly also being a potent source for predicting future capabilities (Mai, 2016).

## 5 Conclusion: Death to Grades, Courses, and Degree Programs?

As the title carries a question mark, this paper is not making a certain case that grades, courses and degree programs as we currently know them, will disappear. Grades may become an anachronism in the modern world of AI and big data analytics. Systems analyzing large amounts of raw data on the fly might give decision makers much more targeted information than static reports (i.e., grade transcripts) that have distilled these raw data according to one particular dimension decided by somebody else. Static reports are not agile because they follow a fixed format, thus are not adaptable to the changing information needs of decision makers. With the increasing ability to quickly analyze detailed records of candidate achievements, as well as test candidates directly in the application process for jobs or admission to further studies, grades may end up being less useful for the documentation of competencies.

For the potential death of courses, the point is that if students are allowed to be agile and choose learning outcomes on a fine-grained level, with strong students doing more learning outcomes in a semester, weaker students doing fewer, the border between one course and the subsequent one, or between one course and another parallel one, will be blurred. This resembles the argument that IT might help facilitate a revival of PSI (Personalized System of Instruction), with its idea of student self-pacing rather than one pace fits all (Eyre, 2007). With learning and assessment supported by highly automated delivery and testing, it may no longer feel necessary to decide exactly which learning outcomes are bundled into one particular course. The university might still present *recommendations* (e.g., these 12 learning outcomes are best pursued together in the same semester), but students might be allowed to go for a different pace, or different decomposition.

On a higher level of aggregation, the same argument could apply for degree programs. If a student selects a little of this and a little of that, and it leads to a nice job, why would that be wrong? If the future sees the implementation of one or more huge, ontology-supported study-job-matching platforms, where students can see what jobs they may be moving towards, and what competencies they may be lacking to get there, it is unlikely that they would make weird choices. Instead, student choices along the way might be more precise than choices made by faculty several years earlier according to what competencies faculty then believed necessary for a successful career.

One possibility is that degree programs will have to become a lot more flexible than they tend to be today—with less rigid requirements for how much students have to learn on various subjects, and in what sequence—to allow for more student agility. Another possibility is that degree programs will gradually disappear, instead allowing each student to emerge with a personal degree, of whatever size and topic mix that fits the job market or individual preferences. Again, universities could maintain degree programs as *recommendations*: This is a package of learning outcomes that we believe qualifies well for a certain set of current and future jobs, and this is what we think is the best pace and order of achieving these learning outcomes.

Some students have predominantly economic motivations, preferring studies with a high likelihood of yielding a well-paid job. If their finances up front are strained, they might not be able to afford years of full-time study. For these students, the ultra-lean approach of micro-degrees might be the best option—identifying a minimum viable competence that can quickly improve their employability. In the future one might envision such an approach worked in numerous iterations: Studying for a short time to get a job, while working saving up some of the money to afford a new brief study period, this leading to a new job with higher salary (or increased salary in the current position), etc. Other students, in a less strained situation, can afford to prioritize their studies more according to personal preferences, going for years of full-time study and changing their minds along the way, aiming at an education with some general competence that can fit many jobs. Neither the rapid pace of micro-degrees, nor changing one's mind along the way, tend to fit today's dominant Big Decision Up Front degree programs. In an ever more fast-paced future, it is reasonable to assume that both students and universities need more incremental and iterative decisions on what to study, and how best to teach it and learn it.

**Acknowledgments** The research for this paper took place in the context of the Excited Centre for Excellent IT Education, funded by NOKUT for the period 2016–21.

## References

- Alp, N. (2001). *The lean transformation model for the education system*. Paper Presented at the Proceedings of the 29th Computers and Industrial Engineering Conference, November.
- Baker, R. S., & Inventado, P. S. (2014). Educational data mining and learning analytics. In *Learning analytics* (pp. 61–75). Springer.
- Balfour, S. P. (2013). Assessing writing in MOOCs: Automated essay scoring and calibrated peer review (tm). *Research & Practice in Assessment*, 8.
- Beck, K., Beedle, M., Van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., ..., Jeffries, R. (2001). Manifesto for agile software development.
- Beetham, H., & Sharpe, R. (2013). An introduction to rethinking pedagogy. In *Rethinking pedagogy for a digital age: Designing for 21st century learning* (pp. 41–61).
- Biesta, G. (2009). Good education in an age of measurement: On the need to reconnect with the question of purpose in education. *Educational Assessment, Evaluation and Accountability (formerly: Journal of Personnel Evaluation in Education)*, 21(1), 33–46.

- Brynjolfsson, E., & McAfee, A. (2014). *The second machine age: Work, progress, and prosperity in a time of brilliant technologies*. WW Norton & Company.
- Cantor, J. A. (1997). Experiential learning in higher education: Linking classroom and community. ERIC Digest.
- Caroleo, F. E., & Pastore, F. (2017). Overeducation at a glance. Determinants and wage effects of the educational mismatch based on AlmaLaurea data. *Social Indicators Research*, 1–34.
- Chaudhuri, S., & Dayal, U. (1997). An overview of data warehousing and OLAP technology. *ACM SIGMOD Record*, 26(1), 65–74.
- Chun, A. (2004). The agile teaching/learning methodology and its e-learning platform. In *Advances in Web-Based Learning–ICWL 2004* (pp. 745–784).
- Clayton, C. (2014, November 2014). Jeff Weiner would like to connect with you on LinkedIn. *Sky*.
- Coppola, G., & O’Higgins, N. (2017). *Youth and the crisis: Unemployment, education and health in Europe*. Routledge.
- Craig, R., & Williams, A. (2015). Data, technology and the great unbundling of higher education. *Educause Review*, 50(5).
- D’Souza, M. J., & Rodrigues, P. (2015). Extreme pedagogy: An agile teaching-learning methodology for engineering education. *Indian Journal of Science and Technology*, 8(9), 828.
- Denton, P., Madden, J., Roberts, M., & Rowe, P. (2008). Students’ response to traditional and computer-assisted formative feedback: A comparative case study. *British Journal of Educational Technology*, 39(3), 486–500.
- Dybå, T., & Dingsøy, T. (2009). What do we know about agile software development? *IEEE Software*, 26(5), 6–9.
- El-Abbassy, A., Muawad, R., & Gaber, A. (2010). Evaluating agile principles in CS education. *International Journal of Computer Science and Network Security*, 10(10), 19–28.
- Ellis, C. (2017). The importance of E-portfolios for effective student-facing learning analytics. In *E-portfolios in higher education* (pp. 35–49). Springer.
- Emiliani, M. (2004). Improving business school courses by applying lean principles and practices. *Quality Assurance in Education*, 12(4), 175–187.
- Emiliani, M. (2005). Using kaizen to improve graduate business school degree programs. *Quality Assurance in Education*, 13(1), 37–52.
- Eyre, H. L. (2007). Keller’s personalized system of instruction: Was it a fleeting fancy or is there a revival on the horizon? *The Behavior Analyst Today*, 8(3), 317.
- Frey, C. B., & Osborne, M. A. (2017). The future of employment: How susceptible are jobs to computerisation? *Technological Forecasting and Social Change*, 114, 254–280.
- Gruber, T. R. (1995). Toward principles for the design of ontologies used for knowledge sharing? *International Journal of Human-Computer Studies*, 43(5–6), 907–928.
- Hakulinen, L., & Malmi, L. (2014). *QR code programming tasks with automated assessment*. Paper presented at the Proceedings of the 2014 Conference on Innovation & Technology in Computer Science Education.
- Hallgren, M., & Olhager, J. (2009). Lean and agile manufacturing: External and internal drivers and performance outcomes. *International Journal of Operations & Production Management*, 29(10), 976–999.
- Holmegaard, H. T., Ulriksen, L. M., & Madsen, L. M. (2014). The process of choosing what to study: A longitudinal study of upper secondary students’ identity work when choosing higher education. *Scandinavian Journal of Educational Research*, 58(1), 21–40.
- Kohn, A. (1994). Grading: The issue is not how but why. *Educational Leadership*, 52(2), 38–41.
- Laband, D. N., & Lentz, B. F. (2003). New estimates of economies of scale and scope in higher education. *Southern Economic Journal*, 172–183.
- Lemoine, P. A., & Richardson, M. D. (2015). Micro-credentials, nano degrees, and digital badges: New credentials for global higher education. *International Journal of Technology and Educational Marketing (IJTEM)*, 5(1), 36–49.
- Mai, J.-E. (2016). Big data privacy: The datafication of personal information. *The Information Society*, 32(3), 192–199.

- Moogk, D. R. (2012). Minimum viable product and the importance of experimentation in technology startups. *Technology Innovation Management Review*, 2(3), 23.
- Moreno Minguez, A. (2013). The employability of young people in Spain: The mismatch between education and employment. *Online Submission*, 3(5), 334–344.
- Nguyen, M. (2012). Degreeless in debt: What happens to borrowers who drop out. Charts you can trust. *Education Sector*.
- Nilsson, S., & Nyström, S. (2013). Adult learning, education, and the labour market in the employability regime. *European Journal for Research on the Education and Learning of Adults*, 4(2), 171–187.
- Nutter, P. W., Pavlidis, V. F., & Pepper, J. (2014). *Efficient teaching of digital design with automated assessment and feedback*. Paper Presented at the 10th European Workshop on Microelectronics Education (EWME).
- O’Keeffe, P. (2013). A sense of belonging: Improving student retention. *College Student Journal*, 47(4), 605–613.
- Poppendieck, M., & Cusumano, M. A. (2012). Lean software development: A tutorial. *IEEE Software*, 29(5), 26–32.
- Rosa, H. (2003). Social acceleration: Ethical and political consequences of a desynchronized high-speed society. *Constellations*, 10(1), 3–33.
- Salomon, G. (2016). It’s not just the tool but the educational rationale that counts. In E. Elstad (Ed.), *Educational technology and polycontextual bridging* (pp. 149–161). Rotterdam: SensePublishers.
- Sokolov, M. (2001). Technology’s impact on society: The issue of mass-customized education. *Technological Forecasting and Social Change*, 68(2), 195–206.
- Srikanth, M., & Asmatulu, R. (2014). Modern cheating techniques, their adverse effects on engineering education and preventions. *International Journal of Mechanical Engineering Education*, 42(2), 129–140.
- Waters, J. K. (2015). How nanodegrees are disrupting higher education. *Campus Technology*, 5.
- Whalen, R., Freeman, S., & Jaeger, B. (2008). Agile education: What we thought we knew about our classes, what we learned, and what we did about it. *Proceedings of the American Society for Engineering Education, Pittsburg, PA*.



# Leveraging Agile Methodology to Transform a University Learning and Teaching Unit



Madelaine-Marie Judd and Heidi Christina Blair

**Abstract** Student diversity and other contextual factors have placed increasing pressure on university academics to work harder, faster and in more innovative ways. Within Australian universities, the central Learning and Teaching Units (LTUs) are charged with the responsibility of improving the quality of education and providing academic development to educators to enhance curriculum, assessment, online education, and any and all other pedagogical matters. The heightened contextual demands that face academics, schools and faculties funnel into whole-of-university pressure on LTUs to prioritise, juggle, leverage and deliver outcomes with, and for, the student-facing academics and divisions. This chapter presents an annotated case study of an Australian LTU in which Agile methodology was successfully applied to achieve whole-of-institution quality improvement. The chapter authors, as members of the LTU team, share the theoretical underpinnings of their practice. The key takeaways are recommendations and strategies for heightening buy-in and active staff engagement, as well as how to clearly and transparently communicate to assure widespread adoption. The surprising outcome of the Agile approach was unexpected gains in confidence and self-efficacy described by numerous participating stakeholders. This chapter unpacks these unanticipated outcomes with retrospective recommendations for readers applying Agile approaches. While this case study features application of Agile methodology through a LTU within an Australian university, the authors have explicitly addressed areas in which recommendations apply to a range of other industries and sectors.

**Keywords** Agile methodology · Higher education · Learning and teaching  
Project management · Organisational transformation · Academic development

---

M.-M. Judd (✉)  
The University of Queensland, Brisbane, Australia  
e-mail: [madelainemariej@gmail.com](mailto:madelainemariej@gmail.com); [m.judd@uq.edu.au](mailto:m.judd@uq.edu.au)

H. C. Blair  
Griffith University, Brisbane, Australia  
e-mail: [h.blair@griffith.edu.au](mailto:h.blair@griffith.edu.au)

© Springer Nature Singapore Pte Ltd. 2019  
D. Parsons and K. MacCallum (eds.), *Agile and Lean Concepts for Teaching and Learning*, [https://doi.org/10.1007/978-981-13-2751-3\\_9](https://doi.org/10.1007/978-981-13-2751-3_9)

## 1 Introduction

Business analysts and researchers alike write that twenty-first-century industries and organisations that fail to be fast-moving, productive, efficient and responsive to change rarely survive (Brown, Holtham, Rich, & Dove, 2015; Craig, 2004; Ross, Ressia, Sander, & Parry, 2017). In Australia, the higher education sector is no exception. There are 41 Australian universities across eight states and territories with a population of over 24 million people (Australian Bureau of Statistics, 2017). In 2016, 1.3 million Australian and international students were enrolled, and 120,000 full-time equivalent staff were employed, across Australian universities (Universities Australia, 2018). Currently, only 39% of 25–34 year-olds in Australia have a bachelor degree or higher (Universities Australia, 2018). However, the Australian government wants to increase that number. In 2012, the Australian government introduced demand-based funding, which meant that the Commonwealth Government would subsidise as many qualified students who applied for whichever study programmes they chose (Dow, 2013; Ross, 2018). The outcome was a sizeable increase in university enrolments (Dow, 2013). The consequences were positive for Australians in that between 2008 and 2016 there was a 106% increase in the number of students with disabling conditions, an 89% increase in Aboriginal and Torres Strait Islander students, a 55% increase in students from families classified within low socio-economic brackets and a 48% increase in the number of students from regional and remote Australia (Universities Australia, 2018).

In this context of increased student enrolment, competition across the 41 universities has heightened as each has ramped-up marketing and articulation of distinctive value-propositions to attract increased student numbers and particularly applicants of higher calibre (Bradmore & Smyrnios, 2009). Accompanying the heightened competition to attract students, and the resultant workloads of staff to accommodate these higher numbers, other forces exerted further pressures on Australian universities. These factors included greater global competition, rising government standards and expectations, rising public expectations, increased comparisons through published university rankings, decreasing public funding, and recognition of the potential of technological disruptions from the growth of computer hardware, software and networking (Currie & Vidovich, 2000; Twidale & Nichols, 2013). This has resulted in an increasing workload on university staff in a context of mounting funding cuts.

In order to support and ensure that academics had access to specialised professional pedagogical development, to manage quality assurance processes, and to inspire and lead innovation, Learning and Teaching Units (LTUs) were opened in each of Australia's universities (Asmar, 2002; Brew & Cahir, 2012). LTUs are described by Holt, Palmer, and Challis (2011) as 'contributing to the growth of staff capabilities for teaching and learning development, innovation and advancement' (p. 7). LTUs have been lauded for being at the leading edge of pedagogical practices enabling university teaching staff to develop learning experiences that leverage innovative design and technology (Kolomitro & Anstey, 2017). These central units play significant roles in enabling universities to enact missions of delivering high quality learning

experiences for students and adapting to the accelerating pace of change facing universities today (Holt et al., 2011; Kolomitro & Anstey, 2017). Consequently, the adoption of new processes within LTU's is critical in order to develop authentic and meaningful learning experiences for students and staff.

This chapter presents a case study of good practice, exploring how one LTU at an Australian university adapted and adopted an approach to managing these challenges to deliver stronger outcomes. This chapter argues that Agile methodology is an effective approach to enhance the self-efficacy of LTU staff, whilst at the same time developing productive working relationships with stakeholders—in the context of this case study, academic stakeholders. The case study was set at Griffith University, an institution with five campuses in Southeast Queensland, with a student population of 50,000, and ranking within the top 3% of universities worldwide (Universities Australia, 2017). The case study is further situated within the LTU, consisting of approximately 50 staff in roles such as web developers, educational designers, project managers (referred to as project leaders), and programme/professional learning consultants. At any given time, the LTU leads approximately 10 whole-of-institution initiatives, that incorporate stakeholders from other central departments (e.g. Careers and Employability, Student Services, etc.) and academic members from each Faculty. The key objectives of the featured LTU include, 'facilitate positive professional identity and capability... build leadership for learning to enable our University to be the best it can be, and [to] stimulate innovation' (Griffith University, n.d.).

Specifically, this case study presents the LTU's application of an adapted Agile methodology. Agile methodology has been defined as 'a set of iterative and incremental software engineering methods' that are used to develop products or resources (Dikert, Paasivaara, & Lassenius, 2016, p. 88). Almost all definitions (including the one quoted here) situate Agile methodology within the context of engineering and ICT. This chapter therefore adapts the definition to expand its applicability to a wider context. Throughout the remainder of this chapter, Agile methodology is defined as, *regular, structured and systematic team-based processes to quickly achieve progressive goals*. Another key term, which is often used synonymously with *agile* but is actually an embedded component of the methodology, is *scrum*. The term *scrum* is a shortened form of the word *scrummage*, which is used in sports like rugby, where the players huddle together in a tight pack with their heads down. Whereas in sports-based scrums, the players are competing against each other in teams, trying to grab hold of the ball, in Agile scrums, the players are working figuratively closely together to ensure that work gets done. Teams often contribute to *scrum meetings* where they briefly report, debrief and collaborate on progress, challenges and communications. Drawing-upon another sports metaphor, *sprints* are a key component of Agile approaches. In this context, it means that work is clustered, or in other words, divided into short goals or sub-goals that can be quickly achieved (e.g., work that can be completed in 3-week bursts). The use of Agile methodology affords development teams the opportunity to co-construct resources or products with user groups, whilst also allowing for flexibility in design (Dingsøyr, Nerur, Balijepally, & Moe, 2012; Serrador & Pinto, 2015). Teams are self-organising, and cooperation between team members and stakeholders are vital components of this methodology (Gren, Torkar,

& Feldt, 2017). This has led to the perception that deploying Agile methodology can enhance user satisfaction as resources have been designed, such that they are informed by user requirements and experiences in an iterative manner (Dikert et al., 2016; Dingsøyr et al., 2012).

The Principles of Agile methodology are one reason for its success. These were adopted and adapted for use in the context of the case study described in this chapter. *Project Rounds* were instances in which the leaders for discrete projects met to share celebrations, updates and challenges and pose questions to the leaders of other projects. These meetings provided a valuable forum in which project leaders could see where projects intersected and consider collaboration between projects. *Production Roundtables* were sessions where all members of the LTU (whether or not they were actively involved on Agile project teams) were invited to share updates on the projects that they were working on and encouraged to identify constraints or issues that they were having. Finally, *Monday Morning Conversations* were an informal coffee hour open to all staff involved in any component of the Agile projects. At these sessions, personal and professional celebrations were shared, questions for consideration were posed, demonstrations were provided and informal reports from conferences were given.

Studies have indicated a positive perception regarding the use of Agile methodology to enhance project (and sometimes overall unit) success (Serrador & Pinto, 2015). In a survey study of over 1,000 projects from a range of disciplines, Serrador and Pinto (2015) analysed the extent to which the use of Agile methodology led to each project's perceived success. It was the overall finding from this study that the use of Agile methodology led to enhanced perceptions of project success in regards to stakeholder satisfaction, efficiency of effort and project outcomes. This is similarly found in the work of Stettina and Hörz (2015) who undertook 30 semi-structured interviews with project managers across 14 European organisations. In these interviews, project managers were asked about their processes and challenges encountered through their projects. A key finding of this study was that participants perceived Agile projects to produce resources that greater aligned with the requirements of their user groups. A second element of success to Agile methodology is that it draws upon the unique strengths and areas of expertise of each team member (Serrador & Pinto, 2015). It thus becomes crucial to foster a culture in which these strengths are identified and team members are empowered to effect change (Gren et al., 2017).

There is currently a paucity in the literature pertaining to the use of Agile methodology within the university sector, and in particular, outlining its applicability for LTUs. This chapter provides an annotated case study of how an Australian LTU implemented an Agile methodology to lead the university in innovative projects. It is asserted in this chapter that implementing a contextualised Agile development methodology can be a powerful strategy for LTUs in meeting the challenges resulting from the rapid rate of change to the sector, and furthermore, that Agile methodology is an efficacious project management approach for contexts beyond engineering and ICT. This chapter explores how to implement a bespoke model and highlights some of the key strategies to enabling cultural buy-in and development.

## 2 Case Study Context

In 2016, the Griffith University LTU developed and implemented a bespoke model of Agile methodology. Prior to this, the LTU had adopted more ‘traditional’ methodologies, such as *waterfall project management*. With the increase in the scope and number of projects, these projects including initiatives focusing on student employability, implementing an ePortfolio system, capabilities-based professional learning, educational design and interactive video projects, the LTU was required to deliver projects faster, with greater stakeholder involvement. Additional conditions inspiring the need for a new approach, were that team members within the LTU needed to manage their time and energy between multiple projects, and the involvement of stakeholders fluctuated depending upon their workload. In addition, a compounding time-squeeze resulting from the university moving from a semester (2 semesters per academic year) to a trimester academic calendar (meaning more teaching weeks in the year and less down-time between teaching periods to action these projects). A distinct need was thus recognised by the LTU leadership team for a shared approach, process and framework for project management that would nurture collaborative relationships, build the LTU’s internal and external reputation, and foster the creation of innovative learning opportunities for students. Following a review of project methodologies, the LTU chose to develop and implement a bespoke Agile methodology. The principles articulated within the Agile Manifesto (Agile Alliance, 2001) were a good fit for the LTU’s staff desire to work more collaboratively to develop projects in a manner that was flexible and enabled iterative releases of learning experiences (Sweeney & Cifuentes, 2010). In addition, key principles such as accountability and teamwork, self-managing, cross-functional teams, and team-based responsibility and sharing all resonated with the distinct needs of the LTU (Scrum Alliance, 2017).

### 2.1 Working Group

One of the key early stages of the process was to establish a Working Group. The Working Group was the equivalent of a committee executive, but without the role division and stratification, in that all members were intentionally equal and the structure flat. This non-hierarchical structure was intentionally designed to reflect the key principles of Agile methodology (Dingsøy et al., 2012; Gren et al., 2017). The Working Group was initially comprised of primarily LTU staff, but expanded as the projects took-hold and brought in members from across the University (this included academic staff from Faculties, and members of Central Units). The initial Working Group developed the guiding principles, which were revisited and used as touch-points throughout projects. The guiding principles are shared verbatim here because they not only provide insight into the deliberations of the Working Group, but more broadly, describe the overall characteristics and scaffolding of the entire application of Agile methodology within a University LTU.

- Any person in any role can lead a project.
- Project Leads are stewards of the project, not task managers.
- Collection of insights/needs (i.e. User Stories) regarding the use of a project's deliverables are gathered from a broad representation of potential user groups.
- All Development Team Members (i.e. staff who dedicate the most hours on each project to create and refine the materials, resources and other outcomes) recognise that flexibility is required during the lifecycle of a project.
- The project embraces iterative development in short development periods (i.e., Sprints) with deliverables at the end of each Sprint.
- Scheduled meetings will be held for viewing the completed work (i.e. Sprint Reviews), through collaborative presentations made by multiple stakeholders and including large-group feedback.
- Transparency across projects enables synergy between projects and project teams.
- Common language for roles, processes and tools is critical.
- Development Team Members will be empowered to determine the tasks to be completed during a given Sprint (i.e. Sprint Planning Meetings).
- Development Teams are formed not only on skills and knowledge needed for a specific project, but also personal interest in the project.
- Stakeholders should be included from each academic group and other central entities as appropriate. A variety of roles should be represented including academics, learning and teaching professionals, support personnel and other key stakeholders.

Over time, shared processes and understandings began to coalesce which included values, principles, processes and vocabulary. Volunteers from the Working Group created a draft framework. A dedicated workshop was then run to engage all members in an exploration of the draft framework and included opportunities to provide feedback—thus reflecting the iterative process design of Agile (Dikert et al., 2016; Dingsøy et al., 2012). Upon confirming that the Working Group's efforts were in alignment with the vision and goals of the LTU leadership team, the Working Group continued to meet to integrate feedback and evolve the framework.

## 2.2 *The Agile Methodology Framework*

With the following foci in mind, the LTU developed a culture in which project leadership was distributed across multi-disciplinary teams; members of project teams self-nominated, as opposed to being assigned to projects; stakeholders were co-constructors of resources, as opposed to clients; and project leaders were stewards, as opposed to managers (Dikert et al., 2016; Dingsøy et al., 2012). These fundamental principles guided the development of our collaborative framework—the *Agile Management of University Projects Framework*—and our resultant interactions with the university community.

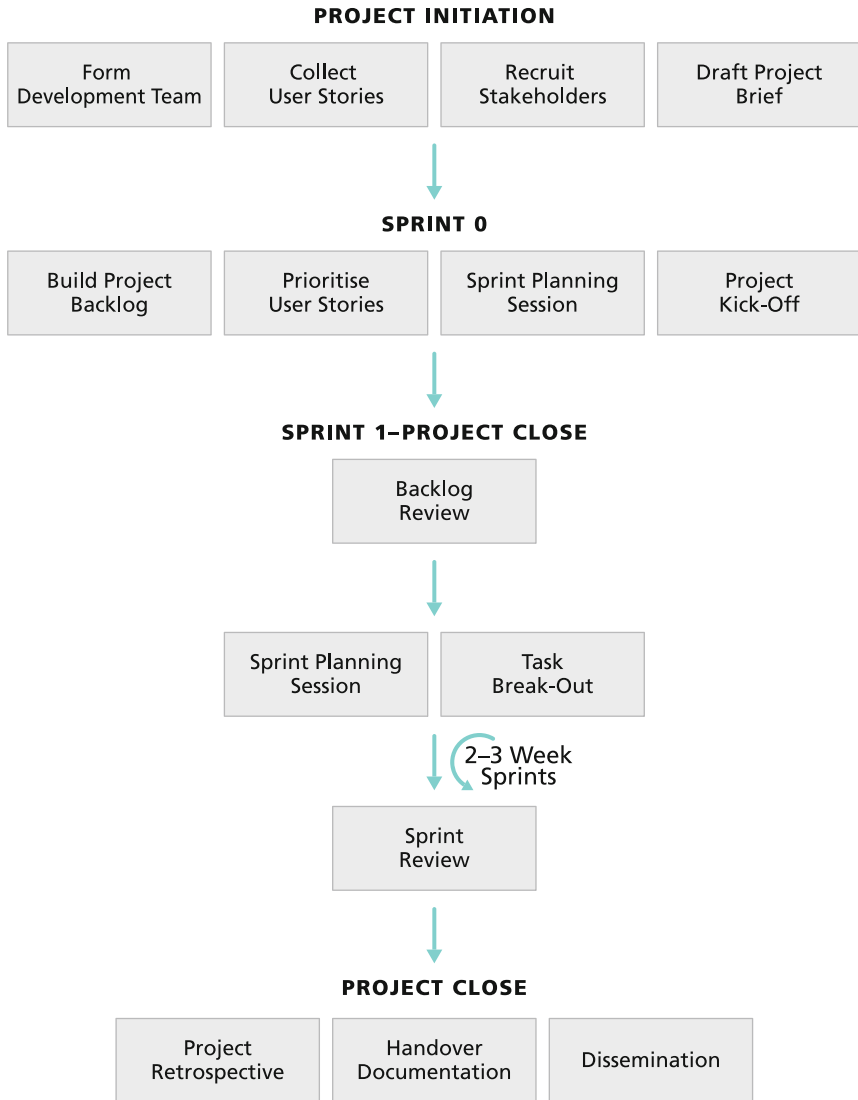
The *Agile Management of University Projects Framework* (Fig. 1) outlines the processes and interactions of the bespoke model. As outlined visually within Fig. 1,

*Project Initiation* constitutes the first phase of any project. In this phase, the Development Team is formed. The Development Team consists of the project owner, the project leader, and in our LTU context educational designers, graphic designers, web developers and/or videographers. After the Development Team has been formed, Stakeholders are recruited, the project brief is drafted and user stories are collected from both the Stakeholders and the Development Team. A user story details the type of user and some key requirements or considerations to inform the resource development. Prompting cards are given to stakeholders which include 'As a \_\_\_ I want \_\_\_ so that I can \_\_\_\_.' In the context of our LTU, an illustrative example included 'As an educator, I want good practice examples of learning activities so that I can align my curricula to accreditation requirements.' Once the steps of *Project Initiation* have been completed, the project will then proceed to the next phase, referred to as *Sprint Zero*. In *Sprint Zero*, the user stories are prioritised by the Project Owner and Project Leader. The Stakeholders, Development Team, Project Owner and Project Leader all convene for the *Project Kick-off* meeting in which group norms and community principles are established, roles/responsibilities are agreed upon and an overview of the bespoke model are discussed at length. Following the *Project Kick-off* meeting, the Development Team then meet for the first *Sprint Planning Session* in which, based upon the prioritised user stories, members self-nominate tasks that they can personally achieve within the first sprint. The prioritised user stories will be the central focus of *Sprint One*.

The number of sprints within a project will depend upon the scope, size and timeframe of the project. In the context of our LTU, project timelines, and thus the resulting number of sprints, are influenced by contextual factors such as trimester timelines and the academic calendar. As depicted within Fig. 1, each sprint follows a similar, circular pattern, represented visually in the framework by the circular arrow. Once the sprint has been completed, the next sprint (and thus the same steps) repeat. Each sprint consists of a Sprint Planning Session and a Sprint Review Meeting. In the planning session, the Development Team self-assigns tasks that support the focus of the next sprint, pulling user stories in from the backlog. At the conclusion of the sprint, the members of the Development Team present their work to project Stakeholders at the Sprint Review Meeting. During this meeting, the project stakeholders review the work completed in the previous sprint and provide feedback. Encouraging Development Team members to present their own work, greater fostered a distributed leadership model in which team members were empowered and recognised for their achievements (Gren et al., 2017). At the conclusion of this meeting, the next sprint commences, thus representing an iterative release cycle, in which small deliverables are released and then amended in-line with stakeholder feedback. The third and final phase is *Project Close*. A key activity during this phase is the meeting of all stakeholders and development team members for a Project Retrospective. In the Project Retrospective, Stakeholders and Development Team Members alike meet to review the entire project, discuss what worked well, what could be improved upon and next steps including dissemination strategies and hand-over to the operational stewards.

Whilst the framework is procedural in nature, the implementation of this framework shaped and transformed the culture of our LTU. Agile methodology accounted

# Agile Management of University Projects Framework



**Fig. 1** Agile Management of University Projects Framework

for the range of contextual factors influencing and impacting upon the delivery of project resources and outcomes, and further fostered a distributed leadership model throughout the LTU. Thus, not only did the framework transform the way in which



we worked as a team, but also the perception of our LTU throughout the broader university community.

### 3 Process of Agile Methodology

A definitional element of Agile methodology that needs to be explicitly articulated, reinforced and demonstrated over and over again, is that it is a generative and co-constructed process that changes alongside the team membership and shifting goals, as milestones are reached. A Senior Curriculum Consultant explained,

Whilst I initially outlined the project vision, aims and goals, the Agile-like methodology and encouraged an inclusive, open and collegial space for us to collaborate, it took a few meetings for everyone involved in the project to realise that we were co-developing the product (story model) and that their ideas were being incorporated into the product. We were all coming to terms with the fact that the project was in a state of evolving and that we were all part of that.

One of the key team members of an Agile project team within the case study described in this chapter reflected on positive reception to the defined process and intentional structures of Agile methodology, which she perceived as a pendulum-swing away from typical university-processes entrenched in long-standing institutional cultures of stability and traditions.

I think working at a university, the cogs often move very slowly, the Agile process provided structure and milestones that got results quicker... the weekly targets made people more accountable and people who were known not to deliver were gently reminded that they were important and the work they were tasked to do was needed for the project to move forward.

As explained in the quote above, Agile methodology was confirmed as an efficacious approach to enabling university staff to change, because it placed the scaffolds and supports in-place that staff members needed to sustain the process and achieve the desired outcomes.

While the overall stance of Agile methodology can seem unnerving to staff used to more traditional processes, structured techniques and strategies provide stabilising footholds to many staff. An Educational Designer described initial workshops as a fundamental element of Agile methodology.

At the beginning of the projects, I held workshops with the wider stakeholder group to collect the user stories. These workshops served to define roles as well as identify users and user requirements. As the project moved onto design and development, the stakeholder group would narrow, but as the project finalised, I was able to build on the relationships fostered at the first workshops to collaboratively implement the solution.

As described above, the content of the initial workshops served as a touchpoint that could be revisited as the process progressed. A Web Developer described the benefit of the scrum approach, as a component of the overall Agile methodology.

Our project team's success was built on a solid foundation of Agile Scrum. The two to three-week sprints allowed for focused development time paired with feedback from our project owner and stakeholder group during reviews. This constant feedback cycle enabled us to always be getting closer to user expectations of how the product should evolve.

It is unsurprising that this quote, about the value of the scrum approach, was articulated by a Web Developer. This approach is a good match for the culture of staff within information technology units as scrum processes enable staff groups to break-work into manageable parts and achieve outcomes along the way. Within the scrums, work is broken into sprints, which ensure that goals are reasonable and achievable. An Educational Designer explained,

Booking sprint reviews every three weeks into the calendars of stakeholders and developers also had an advantage, whereby, commitment to the project was not just communicated but demonstrated, as we made a promise to spend the time together to get the work done.

As he explained, an essential process to ensure success of the projects was to integrate with electronic calendars and other such established work processes to ensure that the work could be incorporated into business-as-usual patterns. A Web Developer emphasised the importance of ongoing communication throughout the work.

From my perspective, this [communication] went really well in our team. For instance, the daily stand-ups helped us to communicate to each team member on a daily basis the state of the development, who is doing what or what element is required by others. This reduced a lot of waiting periods in the developing process.

The benefit of 'daily stand-ups' is that they take very little staff time and effort, but prevent significant problems that might otherwise occur, such as task duplication and unplanned redundancy.

## 4 Agile Methodology Projects Within Organisational Culture

One of the principles that needs to be continuously reinforced for the successful coordination of Agile projects is the recognition that while for the central managing unit the endeavor might seem like the highest organisational priority, stakeholders on the fringes might see the work as one of many competing demands on their time, energy and resources. For example, a Deputy Head of School commented, '*My only disappointment with this project was my inability to participate on a more regular basis due to other commitments.*' Given these time constraints, it is therefore essential that Project Leaders, and other key staff, maintain process flexibility and take an inclusive and transparent approach to communication, providing information, resources and supports in a manner that works best for those who are participating (Dingsøyr et al., 2012; Serrador & Pinto, 2015). A specific example came from an Academic, whose responsibilities as Deputy Dean meant that he was unable to

attend the project meetings. A Project Leader therefore met with him one-on-one to consult and collaborate on necessary decisions in the process. At the conclusion of the process, he provided feedback that, *'Flexibility to meet one-on-one when you couldn't make meetings was highly beneficial. There was clear evidence in following Sprint Reviews that your voice had been heard and considered.'* His comments that he felt listened-to was affirming in a process that sought to be collaborative and inclusive.

## 5 Professional Learning Needs

Professional learning needs of LTU and other staff emerged as Agile projects progressed and tools were adopted. Members developed expertise in tools and processes leading to the creation of support resources and the facilitation of hands-on workshops and coaching. These supports became organic and synergistic, in that members of project teams who became experienced in particular components provided coaching to other teams when they reached the point that they needed these just-in-time skills, strategies and resources. A website was created to host these materials to provide easy access and distribute resource management. For many staff serving as Project Leaders, the website provided a new lens through which to view their work. For example, Senior Curriculum Consultants who traditionally worked with one another to design, create and implement professional learning workshops were now leading Development Teams with a diversity of roles (e.g., Educational Designers, Videographers and Web Developers).

## 6 Unexpected Gains in Staff Self-confidence

One of the key emergent themes across the written feedback from Development Team Members and Stakeholders was gains in staff self-confidence. This was apparent for both those administrating and leading the projects and the academics who were touched-by and contributed to the projects, but not a part of the Development Team or LTU. For example, a Lecturer said, *'Participating in the projects highlighted what I was capable of doing.'* This finding from the context of our LTU extends the work of Serrador and Pinto (2015), in that not only was there an increase in stakeholder satisfaction, but also stakeholder self-efficacy. Furthermore, she stated that she would be using the project outcomes as evidence in her annual performance review. One of the Project Leaders described the benefits (unanticipated but welcomed) in regard to staff self-initiative.

The process empowered the team to take ownership of their role. It wasn't about me speaking for them, it was about them owning their tasks and getting the clarification they needed to continue their work. ... They felt valued and more engaged with the process.

The boost that participation in Agile methodology provided to numerous staff members' self-concepts and transformed the work from a series of time-limited projects, to a sustainable exercise in professional development. Another Project Leader explained that a learning experience, for both staff and those who supervised them, was self-trust.

Let project team staff take control of the work they are doing for a sprint, but ensure they know that you are available for them to call upon if they hit any problems (waiting on others, technology not playing well, unexpected governance).

She explained that within this flat-hierarchy approach, letting-go-of control (or in other words, not micro-managing) was a key learning outcome for staff who were normally in supervisory roles, and furthermore, that this change to overall work culture sustained beyond the end of the Agile projects. Notably, as this application of Agile methodology was conducted with a university context, stakeholders made comments, not only about how they would continue to apply what they learned about Agile methodology to continuous improvement of the functioning of the university as an organisation, but would also use the lessons as curriculum and examples to teach to students (i.e. future Project Leaders). For example, a Deputy Head of School wrote, "*In my role as convener of the course, I can (and will) use the project management skills and processes I have witnessed and experienced as a project stakeholder in this project, as an exemplar in my course.*"

## 7 Implications for Practice

Four top recommendations to consider when implementing a bespoke Agile framework, with a recommendation proposed for each project phase of the framework, are provided in this section. These recommendations are informed by lessons learned and are intended for consideration in a variety of organisations or teams, irrespective of industry or discipline. The first recommendation relates to *Project Initiation* and is to ensure that buy-in of the framework is obtained not only from your Development Team but also your Stakeholders. A key component of this recommendation is to clearly articulate the framework process, co-develop a common understanding of the guiding principles and values, and model agility and flexibility in your approach. In circumstances whereby Stakeholders or Development Team Members cannot attend a meeting, recommend for a proxy to be sent on their behalf, or meet with them individually to ensure their perspectives are heard.

The second recommendation, which relates to *Sprint Zero*, is to seek common agreement on a workflow management tool in which all user stories, tasks and team collaboration will be hosted. Ideally this workflow tool should be transparent, enabling all development team members to incorporate individual tasks, highlight task dependencies and collaborate through comment threads. In the context of our LTU, teams adopted the use of Asana, a web-based workflow management system, to host all project-related information and tasks. This enabled team members to self-

assign tasks during Sprint Planning Sessions, and update the team space throughout the sprints.

The third recommendation, which relates to *Sprint One*, is to celebrate the achievements of your team, and encourage and empower Development Team Members to individually present their work in the Sprint Review Meetings, rather than having the Project Leader present on their behalf. By having development team members individually present, recognises their vast contributions to the project and distributes leadership throughout the team.

The fourth and final recommendation pertains to the *Project Close* phase, specifically the Project Retrospective. Prior to the meeting, ensure that all Development Team Members and Stakeholders are briefed on the purpose of the meeting. Use the meeting as an opportunity to ascertain feedback on how the *Agile Management of University Projects Framework* could be improved and key strengths of the framework to ensure that it continues to be modelled. Take comprehensive notes of feedback provided and develop a report based on these lessons learned. Disseminate these findings to other project teams, so that they can obtain a clearer understanding of what works well, and what can be improved upon.

These recommendations are intended to provide food for thought when developing and implementing a bespoke Agile framework within a team or organisation. By no means are these recommendations exhaustive, but rather have sought to provide some helpful hints when transforming an organisational culture in adopting an Agile framework.

## 8 Conclusion

This chapter took a case study based approach to addressing the strengths, limitations and processual factors of an application of Agile methodology. The case context was an Australian university. The Learning and Teaching Unit (LTU) led multiple whole-of-university projects whereby stakeholders across the organisation collaborated through Agile methodology. Assumptions were confirmed in that the Agile methodology produced strong outcomes and strengthened interaction and cohesiveness between diverse staff and units across the university. Furthermore, this project management approach increased the regard of the LTU. The unanticipated finding was that participation in Agile methodology had a significant and sustainable positive impact on team members' self-confidence. The main contribution of this chapter to published literature is that the application context is outside of management and ICT, which are the dominant disciplines wherein almost every account of Agile methodology is published. The strength of this particular inquiry is that the case study approach enabled deep discovery, including what worked and did not work, and from the experiences of diverse stakeholders. For further inquiry into Agile methodology, researchers are encouraged to build-upon this deep case study inquiry to conduct a broader empirical study with a larger sample size to further explore the primary themes and outcomes of this study.

**Acknowledgements** We would like to thank the following implementers that shared their insights into the development and implementation of the Framework within their work: Dr. Kerry Bodle, Paul Brown, Megan Duffy, Oleg Estrin, Dale Hansen, Darcelle Hinze, Madelaine-Marie Judd, Shar Keshi, Associate Professor Christopher Klopper, Evelyn Lugiarto, Louise Maddock, Simone Poulson, Daniel Tedman, Jeff Teng and Associate Professor Michelle Whitford. Without their work there would be no story to tell.

## References

- Agile Alliance. (2001). *Manifesto for agile software development*. Retrieved from <http://agilemani.festo.org/principles.html>.
- Asmar, C. (2002). Strategies to enhance learning and teaching in a research-extensive university. *International Journal for Academic Development*, 7(1), 18–30. <https://doi.org/10.1080/13601440210156448>.
- Australian Bureau of Statistics. (2017). *Australian demographic statistics, June 2017*. Retrieved from <http://www.abs.gov.au/AUSSTATS/abs@.nsf/mf/3101.0>.
- Bradmore, D. J., & Smyrnios, K. X. (2009). The writing on the wall: Responses of Australian public universities to competition in global higher education. *Higher Education Research and Development*, 28(5), 495–508. <https://doi.org/10.1080/07294360903161154>.
- Brew, A., & Cahir, J. (2012). Achieving sustainability in learning and teaching initiatives. *International Journal for Academic Development*, 19(4), 341–352. <https://doi.org/10.1080/1360144X.2013.848360>.
- Brown, A., Holtham, C., Rich, M., & Dove, A. (2015). Twenty-first century managers and intuition: An exploratory example of pedagogic change for business undergraduates. *Decision Sciences Journal of Innovative Education*, 13(3), 349–375. <https://doi.org/10.1111/dsji.12066>.
- Craig, C. M. (2004). Higher education culture and organizational change in the 21st century. *The Community College Enterprise*, 10(1), 79–89.
- Currie, J., & Vidovich, L. (2000). Privatization and competition policies for Australian universities. *International Journal of Educational Development*, 20(2), 135–151. [https://doi.org/10.1016/S0738-0593\(99\)00065-6](https://doi.org/10.1016/S0738-0593(99)00065-6).
- Dikert, K., Paasivaara, M., & Lassenius, C. (2016). Challenges and success factors for large-scale agile transformations: A systematic literature review. *The Journal of Systems and Software*, 119, 87–108. <https://doi.org/10.1016/j.jss.2016.06.013>.
- Dingsøy, T., Nerur, S., Balijepally, V., & Moe, N. B. (2012). A decade of agile methodologies: Towards explaining agile software development. *The Journal of Systems and Software*, 85(6), 1213–1221. <https://doi.org/10.1016/j.jss.2012.02.033>.
- Dow, C. (2013). Higher education: Sustainability of a demand-drive system. *Parliament of Australia*. Retrieved from [https://www.aph.gov.au/About\\_Parliament/Parliamentary\\_Departments/Parliamentary\\_Library/pubs/BriefingBook44p/HigherEducation](https://www.aph.gov.au/About_Parliament/Parliamentary_Departments/Parliamentary_Library/pubs/BriefingBook44p/HigherEducation).
- Gren, L., Torkar, R., & Feldt, R. (2017). Group development and group maturity when building agile teams: A qualitative and quantitative investigation at eight large companies. *The Journal of Systems and Software*, 124, 104–119. <https://doi.org/10.1016/j.jss.2016.11.024>.
- Griffith University. (n.d.). *Learning futures*. Retrieved from <https://www2.griffith.edu.au/learning-futures>.
- Holt, D., Palmer, S., & Challis, D. (2011). Changing perspectives: Teaching and learning centers' strategic contributions to academic development in Australian higher education. *International Journal for Academic Development*, 16(1), 5–17. <https://doi.org/10.1080/1360144X.2011.546211>.

- Kolomitro, K., & Anstey, L. M. (2017). A survey on evaluation practices in teaching and learning centres. *International Journal for Academic Development*, 22(3), 186–198. <https://doi.org/10.1080/1360144X.2017.1313162>.
- Ross, J. (2018, February 27). Australian universities fight to save demand-driven system. *Times Higher Education*. Retrieved from <https://www.timeshighereducation.com/news/australian-universities-fight-save-demand-driven-system>.
- Ross, P. K., Ressia, S., Sander, E. J., & Parry, E. (2017). *Work in the 21st century: How do I log on?*. Bingley, BD: Emerald Publishing Limited.
- Scrum Alliance. (2017). *Learn about SCRUM*. Retrieved from <https://www.SCRUMalliance.org/why-SCRUM>.
- Serrador, P., & Pinto, J. K. (2015). Does Agile work? A quantitative analysis of agile project success. *International Journal of Project Management*, 33(5), 1040–1051. <https://doi.org/10.1016/j.ijproman.2015.01.006>.
- Stettina, C. J., & Hörz, J. (2015). Agile portfolio management: An empirical perspective on the practice in use. *International Journal of Project Management*, 33(1), 140–152. <https://doi.org/10.1016/j.ijproman.2014.03.008>.
- Sweeney, D. S., & Cifuentes, L. (2010). Using agile project management to enhance the performance of instructional design teams. *Educational Technology*, 50(4), 34–41.
- Twidale, M. B., & Nichols, D. M. (2013). Agile methods for agile universities. In T. Besley & M. A. Peters (Eds.), *Re-imagining the creative university for the 21st century* (pp. 27–48). Rotterdam: Sense Publishers.
- Universities Australia. (2017, March 17). *Griffith University*. Retrieved from <https://www.universitiesaustralia.edu.au/australias-universities/university-profiles/Griffith-University#.Wp58WkqWbIU>.
- Universities Australia. (2018). *Data Snapshot 2018*. Retrieved from <https://www.universitiesaustralia.edu.au/australias-universities/key-facts-and-data#.WvOCGi5uapo>.

# Lean and Agile Assessment Workflows



Michael Striewe

**Abstract** The chapter presents and discusses a structured approach to assessment planning and organization. The approach is inspired by Kanban-style notations as well as by the SEMAT approach for agile software engineering processes. It is based on a breakdown of assessments into their essential elements and phases. It is hence more suitable for agile assessment planning than traditional workflow and process models. It allows to define assessment workflows very easily, so that teachers, item authors and staff can focus better on their individual duties. The chapter is organized in three main sections: The first section introduces the breakdown of assessments into their essential elements and phases. The second section demonstrates how to use these elements in a Kanban-style notation to formulate assessment scenarios. It also discusses examples from practical experience in different scenarios. The third section very briefly elaborates on tool support.

**Keywords** Assessment · Process · Workflow · Planning · Kanban · Phases

## 1 Introduction

Preparing and conducting educational assessments is not an easy thing. First of all, the contents have to be created carefully to make sure that the assessment considers right what is right and wrong what is wrong. Second, test pedagogy and psychometrics have to be considered to make sure that the assessment really measures what it is supposed to measure. Finally, a lot of organizational aspects concerning the when and where have to be considered. Depending on the setting of the assessment, this may involve communication with assessment authorities in case of formal assessments, set-up and management of electronic assessment tools in case of computer-assisted assessments, but also simple communication with participants that has to happen even in informal low-tech assessment scenarios. Consequently, educators will follow some

---

M. Striewe (✉)  
University of Duisburg-Essen, Essen, Germany  
e-mail: [michael.striewe@paluno.uni-due.de](mailto:michael.striewe@paluno.uni-due.de)

© Springer Nature Singapore Pte Ltd. 2019  
D. Parsons and K. MacCallum (eds.), *Agile and Lean Concepts for Teaching and Learning*, [https://doi.org/10.1007/978-981-13-2751-3\\_10](https://doi.org/10.1007/978-981-13-2751-3_10)



kind of workflow even in very simple cases to make sure that the right things happen in the right order (Reynolds, Livingston, & Willson, 2009; Johnson & Johnson, 2002; Banta & Palomba, 2015; Dick, Carey, & Carey, 2014). One can even dig deeper into the formal definition of workflows and processes and notice that training and instruction are sometimes understood as a system which follows not only a process definition, but also has input and output (Laird, Holton, & Naquin, 2003). Since these are classical terms from development and production, one can also think of adapting more terms from that domain, such as lean and agile processes.

In the particular context of technology-enhanced assessment, there was a tendency in recent years to create models of assessment processes that were very detailed and formal. An example for this is presented in Danson, Dawson, and Baseley (2001), which is related to a university-wide process, but limited to a particular existing tool. We can identify roles like 'Exam Office' or 'Students' in conjunction with their activities in these models. The FREMA (Framework Reference Model for Assessment) project collected a more complete set of elements that might occur in process models. One of the project's outputs is a concept map for e-learning assessment processes (Millard et al., 2006; Wills et al., 2007). This map lists activities that turned out to be relevant based on interviews within the assessment community. It covers several didactic aspects such as authoring of assessment items, checking solutions for plagiarism or creating feedback to students, but also organizational issues such as checking the availability of candidates and staff or preparing digital and physical environments. This concept map for e-learning assessment processes is a valuable source of didactic and organizational activities related to assessments, but does not provide an actual technique for modelling actual workflows.

Anyway, restricting educational assessments to strict process definitions following some formal guidelines does not match the daily experience and requirements of educators. There is not the single ideal workflow for educational assessments and there is also not the single ideal workflow for a particular assessment scenario. Instead, educators have to amend and adapt their workflows based on the resources available, the time frame they have to prepare the assessment, and also the number of participants expected to take part in the assessment. Educators also usually do not want to care for a large process description in case of small assessments, but are satisfied with lean descriptions covering the essential elements of the assessment workflow. These essential building blocks may be customized for a particular assessment situation or tailored in the way they are used. These requirements towards an appropriate workflow description can be summarized in a small list of principles

- Workflow elements should be concise and represent reusable steps. This is a value known from lean product development.
- Workflow descriptions should only contain the necessary elements while unnecessary steps are eliminated. This is a value known from lean production.
- Workflow descriptions must be changeable and adaptable. This is a value known from agile development.

There is currently no approach documented in literature that covers all these requirements. However, these requirements represent an understanding of lean and

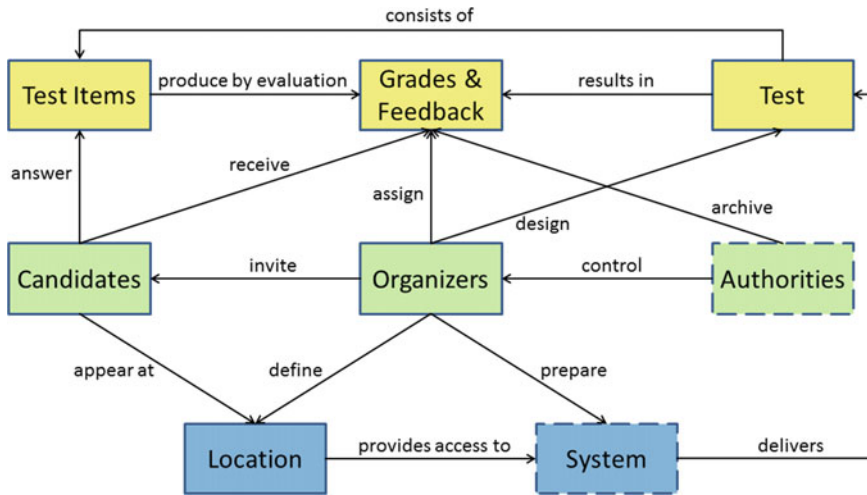
agile workflows that is not specific to educational assessments. Hence, techniques and standards can be reused in this domain that have proven their value already in other domains in which workflows play a major role.

The ESSENCE standard (Essence—Kernel and Language for Software Engineering Methods, 2015) is a modelling standard created by the ‘Software Engineering Methods and Theory’ (SEMAT) initiative and issued by the ‘Object Management Group’ (OMG). It tackles the very same topic of lean workflow descriptions for the domain of software engineering. It defines a modelling language for software engineering process descriptions and a so-called kernel of key elements (named ‘alphas’ and ‘activity spaces’). These are supposed to be relevant in any software engineering project. Each alpha defines a set of states with checklists that allow to track project progress. It is possible to create simple process descriptions by grouping states across alphas and thus defining phases or milestones. It is possible to add more details by assigning ‘work products’ to alpha states or ‘activities’ to activity spaces. This allows a very lean and agile style of defining and using workflows with exactly as much detail and formalisms as necessary (Jacobson, Spence, & Ng, 2013). As a means of graphical representation, the ESSENCE standard introduces the notion of alpha state cards. They are concise representations of an alpha state and its checklist items that can actually be used in the form of small physical cards. These allow enacting many agile practices in planning and monitoring of workflows in a very smooth and natural manner.

While the standard explicitly talks about software engineering, there is no reason to refrain from using the concepts of kernel and alphas in other domains as well. Thus, this chapter uses the ideas of ESSENCE to create both a kernel of key concepts related to educational assessments and some sample workflow descriptions based on this kernel. These descriptions are intended to serve as a blueprint for different workflow descriptions covering several kinds of educational assessments, such as traditional written exams, modern electronic assessments, and oral assessments as well as less formal assessments. A short section on electronic tool support is also included towards the end of the chapter. Hence, the reader may get two main contributions from this chapter: First, it creates a general and unified process model in the domain of educational assessment that covers both traditional and modern forms of assessment. Second, it demonstrates how to define lean and agile workflows on top of this kernel. It thus provides a starting point for modelling one’s own workflows.

## 2 Kernel for Educational Assessment

The ESSENCE *Kernel for Educational Assessment* presented in this section is intended to form a common base for all kinds of educational assessment processes. It is neither limited to a particular didactic purpose of the assessment (e.g. diagnostic, formative or summative) nor to a particular form of assessment (e.g. written assessment, oral assessment or electronic assessment). In order to achieve full but flexible coverage of all kinds of processes in educational assessment, the kernel consists of



**Fig. 1** Overview of the eight alphas in the *Kernel for Educational Assessment* and some of their relationships. Dashed borders indicate optional alphas

eight alphas from which two are optional in most cases. The alphas can be grouped into three so-called ‘areas of concern’ similar to the original ESSENCE standard. An overview of the kernel alphas and some of their relationships is shown in Fig. 1.

## 2.1 Area of Concern ‘Content’

Probably the most important parts of an assessment are its functional contents. They reflect its professional or scientific domain and are represented by this area of concern. Typically, experts in the particular domain create and maintain these contents and assure that they are right. Failure in reaching the desired quality of content in an assessment most likely causes useless assessment results. This area of concern consists of three alphas, representing the different bits an assessment and its results are composed of.

### 2.1.1 Alpha ‘Test Items’

A test item is the smallest consistent unit within an assessment that allows candidates to demonstrate their competencies. For the purpose of this kernel, it is assumed that a test item contains some kind of task description and that the candidates are expected to respond to it in some way, e.g. by ticking answer options, drawing a diagram or answering orally.

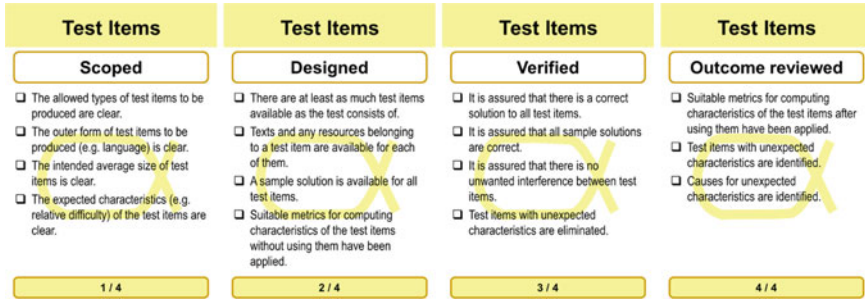


Fig. 2 Alpha states and checklists for the four states of alpha ‘Test Items’

The alpha ‘Test Items’ covers all items potentially used in the assessment and does not ask how an actual test is composed. The test items may form a general item pool or several distinct item pools from which a certain number of items is used in the actual assessment. However, it is assumed that all test items that are potentially used need to be prepared in the same way.

The proposed kernel defines an alpha with four states to represent all essential aspects of test items (see Fig. 2). The names of the first three states are ‘Scoped’, ‘Designed’ and ‘Verified’. They are concerned with the different stages of preparation for test items. The alpha particularly reflects the observation that test items have some formal properties (such as an item type, language and intended difficulty) which are defined in the first state, while their functional properties (such as a task description and a sample solution) are defined in the second state. As legal regulations may explicitly require a second author to do a review of all proposed test items, the third state handles verification and double-checking. The name of the fourth state is ‘Outcome reviewed’. It reflects the didactic practice to review the outcomes of a test with respect to test item performance in order to identify test items with unexpected results (e.g. ones that were often answered wrong by good candidates or ones that were answered right by anybody).

### 2.1.2 Alpha ‘Test’

A test is the actual collection of test items that is delivered to the candidates of the assessment in some way, e.g. by handing out papers, displaying on a screen or asking questions orally. The alpha refers to the test as an abstract construct and hence does not ask whether a candidate actually sees the whole test at once or only can see and answer the test items within the test one after another. There is also no assumption made on whether the test is a static composition of test items or generated adaptively like in computer adaptive testing. Consequently, a test may be the same for all candidates or may be composed individually from one or more item pools.

The proposed kernel defines an alpha with five states to represent all essential aspects of tests (see Fig. 3). The names of the first and second state are ‘Goals

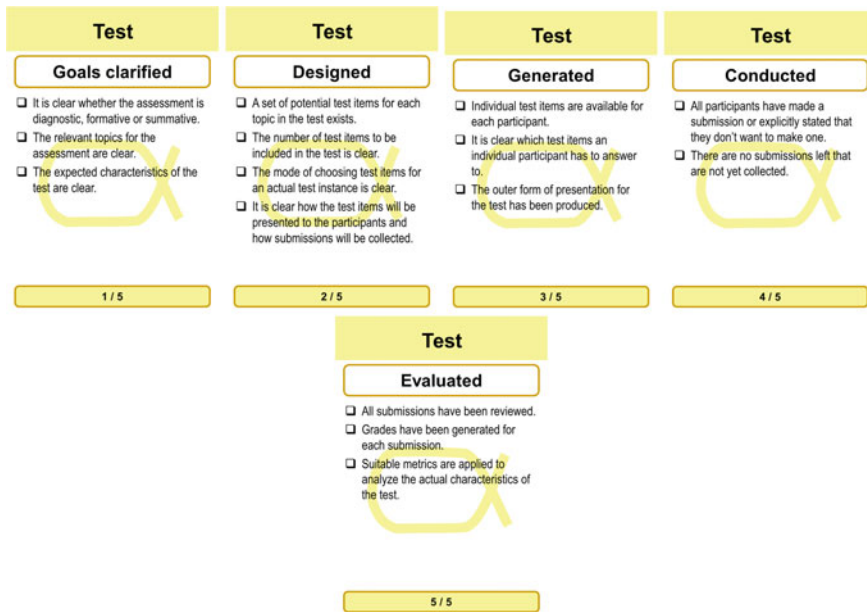


Fig. 3 Alpha states and checklists for the five states of alpha ‘Test’

clarified’ and ‘Designed’. They correspond to the first two states of the alpha for test items, as also the whole test needs both a definition of its formal and functional properties. The name of the third state is ‘Generated’. It is fulfilled when an actual instance of the test is created for each candidate. As already mentioned above, this may be a physical representation such as some pieces of paper, but it may also be the specific sequence of questions asked to one particular candidate in an oral exam. The name of the fourth state is ‘Conducted’. It is fulfilled when all candidates have completed their tests. Notably, in a written exam this state may be reached days or even weeks after ‘Generated’ depending on how long before the day of the test the exam sheets are printed. Different to that it may be reached minutes or even seconds after the last question is posed in an oral exam. The fifth state with name ‘Evaluated’ represents the fact that a test needs to be evaluated. This also includes the retrospective analysis of test item performance as above.

### 2.1.3 Alpha ‘Grades and Feedback’

As the outcome of test evaluation can be very different depending on the didactic purpose and context of an assessment, it is worth modelling grades and feedback as a separate alpha. Each response to a test item contributes to the actual test result, which may consist of marks, credit points, texts, or anything else that is used to inform the candidates about their performance. Results can be assigned both to single test items

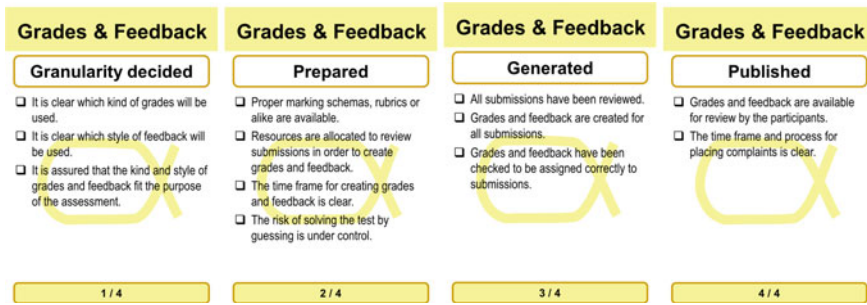


Fig. 4 Alpha states and checklists for the four states of alpha ‘Grades and Feedback’

and to the whole test (or arbitrary parts of it). The alpha covers all these different kinds of feedback and makes no assumptions about whether candidates have access to results during the assessment or only afterwards.

The proposed kernel defines an alpha with four states to represent all essential aspects of grades and feedback (see Fig. 4). Again, the first two are concerned with preparations: State ‘Granularity decided’ reflects that fact that there are many ways of how to give feedback and that the didactic purpose of the assessment determines the choice. State ‘Prepared’ refers to the creation of appropriate marking schemes or alike as well as organizational set-up of grading sessions. The name of the third state is ‘Generated’. It is fulfilled if all grades and feedback are created. The final state is fulfilled when grades and feedback are available to the candidates and is thus named ‘Published’. Notably, in a written exam it may take some time after the submission to reach state ‘Generated’ and it may also take some more time to reach ‘Published’. Different to that, feedback in an oral exam is often generated right after a candidate answered a question and is also published immediately by responding to the candidate’s answer. However, as the alpha refers to grades and feedback in general, state ‘Published’ may nevertheless be fulfilled later, as grades are typically not mentioned after every answer, but only at the end of an exam or even at some later point in time.

## 2.2 Area of Concern ‘People’

Although we already mentioned domain experts as the authors of assessment content, they are not the people in the focus of an assessment for two reasons: First, assessments can be conducted by using predefined tests or test items, keeping the authors completely out of the process. Second, the steps performed by test item or test authors may be domain-specific and are thus out of the scope of a generic kernel for educational assessments.

Hence, this area of concern focuses on people who are more directly concerned with an assessment. It represents each group with a separate alpha: The organizers running the assessment (who may also author test items as part of their duties while preparing the assessment), the candidates taking part in the assessment and optionally the authorities concerned with the legal aspects of the assessment. If either of these parties fails to fulfil their role within the assessment process, there is no guarantee that it will produce the desired outcome.

### 2.2.1 Alpha ‘Organizers’

For each assessment, there is at least one person responsible for organizing it and thus managing the assessment process. For larger assessments, it can be assumed that more people are involved in setting up and conducting the assessment, including test item authors, assessors and technical staff. Each of them pick up parts of the responsibility for conducting the assessment and are thus responsible for some part of the assessment process.

The proposed kernel defines an alpha with four states to represent all essential aspects of the organizers’ duties (see Fig. 5). The name of the first state is ‘Identified’. It represents the fact that it may require some work to find out who needs to be involved into the assessment for which tasks. The name of the second state is ‘Working’. It is fulfilled when all responsible persons have picked up their duties. Once they have done everything that is required to start the actual assessment, state ‘Satisfied for Start’ is reached. Similarly, the final state ‘Satisfied for Closing’ is reached when all evaluation and post-processing is done and the organizers have no more open duties.

### 2.2.2 Alpha ‘Candidates’

The largest group of people concerned with an assessment are usually the candidates, which are the persons who take part in the assessment by solving a test. Although

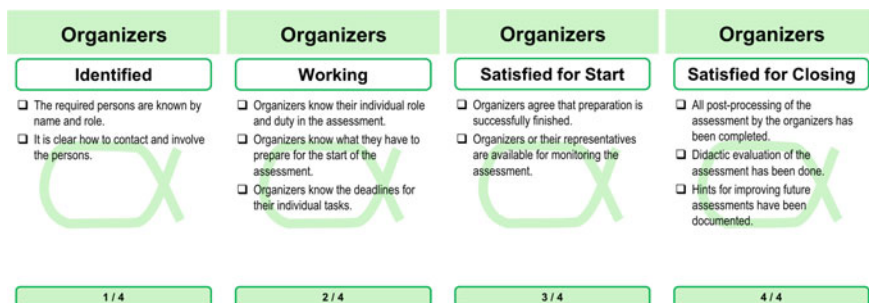


Fig. 5 Alpha states and checklists for the four states of alpha ‘Organizers’

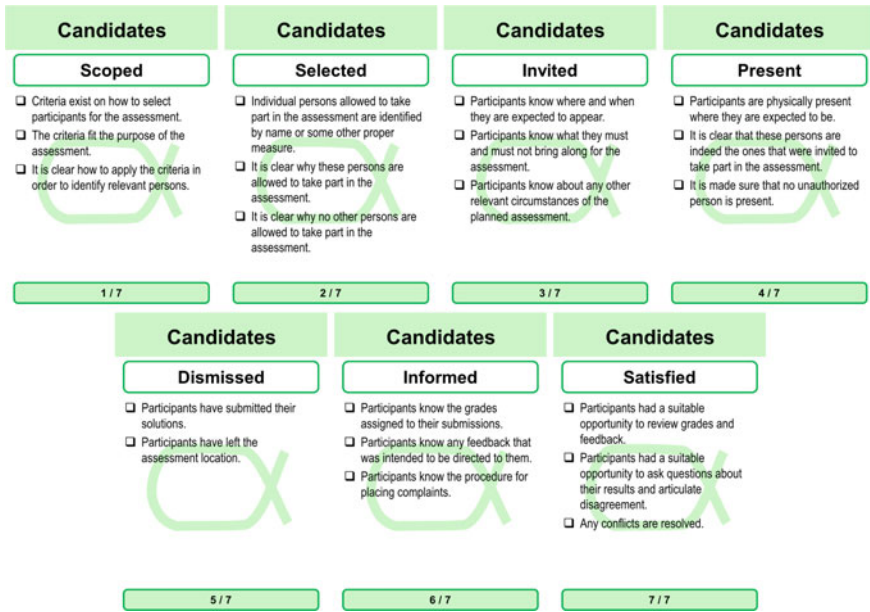


Fig. 6 Alpha states and checklists for the seven states of alpha ‘Candidates’

they are involved personally in the assessment process for a relatively short period of time, the proposed kernel includes an alpha with seven states to represent all essential aspects related to candidates (see Fig. 6). The names of the first two states are ‘Scoped’ and ‘Selected’. They refer to the part of the process in which it is first defined who is allowed to take part in the assessment and second the actual persons are identified. The third state ‘Invited’ is fulfilled when candidates know how to prepare themselves for the assessment. The following two states ‘Present’ and ‘Dismissed’ refer to the physical presence of the candidate at the location where the assessment takes place. Notably, that does not mean that all candidates will be at the same place at the same point in time. They are also considered ‘Present’ if they are in different locations. It is also possible that some candidates are already dismissed, before the last one is present, as it is usual in oral exams. The names of the sixth and seventh state are ‘Informed’ and ‘Satisfied’. They reflect the fact that candidates need explicitly to be informed about their results, which corresponds to state ‘Published’ for grades and feedback. In addition, they also often have some time frame to place complaints before the grades formally count as accepted.

### 2.2.3 Optional Alpha ‘Authorities’

Depending on the didactic and formal setting of the assessment, some official party may be formally responsible for any legal issues related to conducting the assess-



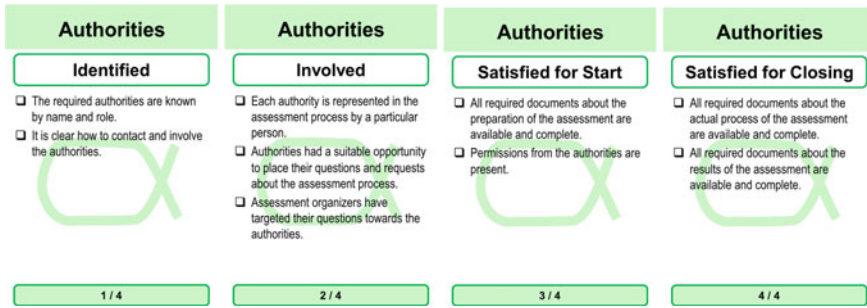


Fig. 7 Alpha states and checklists for the four states of alpha ‘Authorities’

ment. As this may introduce additional process steps or dependencies between states, authorities are introduced as an additional optional alpha in the kernel. This alpha is only relevant for formal assessments. States and checkpoints for alpha ‘Authorities’ are shown in Fig. 7. The name of the first state is ‘Identified’. It covers the same aspects as the corresponding state of alpha ‘Organizers’. The name of the second state is ‘Involved’. It is fulfilled when all assessment information relevant to the authorities have been provided. The naming of the state is different from the second state of alpha ‘Organizers’, as authorities are supposed to play a less active role in the assessment process. Hence they may be involved in terms of providing information or verifying documents, but do not necessarily work in terms of creating contents or making design decisions. The names of the third and fourth state are ‘Satisfied for Start’ and ‘Satisfied for Closing’. This is again similar to the states of alpha ‘Organizers’. They are reached when there are no more legal obstacles to start the assessment or the legal files for the assessment are ready to be closed, respectively.

### 2.3 Area of Concern ‘Logistics’

Besides contents and people, there is also a demand for physical or technical facilities to conduct an assessment. In any case, there are one or many physical locations where candidates are located while taking the assessment. Optionally, they are also using some technical system in case of a computer-aided assessment. This area of concern thus consists of two alphas for the physical and technical aspects of assessment organization. One can imagine adding a third optional alpha for materials or equipment needed during the assessment in case the candidates have to perform physical experiments in natural sciences, artistic or musical exercises using instruments or requisites, or similar. However, the states and checkpoints necessary for this kind of alpha are very likely to be domain-specific. Thus, they are out of scope

for a generic kernel. Instead, they can be added as domain-specific extensions to the kernel, similar to the domain-specific extensions that are defined in the ESSENCE standard.

### 2.3.1 Alpha ‘Location’

It is assumed that each assessment needs some physical location where candidates will be located while taking part in the assessment. Depending on the kind and size of assessment, this may be a single room for all candidates (at the same time or one single candidate or group after another) or a set of distributed locations.

The proposed kernel defines an alpha with six states to represent all essential aspects related to the assessment location (see Fig. 8). Quite similar to the states for candidates, the names of the first two states for the location are ‘Defined’ and ‘Selected’. They refer to the fact that first some abstract requirements are formulated towards the properties of the assessment location and then an actual room or set of rooms is selected. As rooms are physical resources, they may cause conflicts with other assessments happening at the same time. Hence, state ‘Reserved’ is explicitly introduced to cover the necessary communication as well as the calculation of set-up time. If all set-up is done, the location is considered ‘Prepared’, which is the fourth state (corresponding to ‘Satisfied for Start’ for the organizers). The names of the final two states are ‘In Use’ and ‘Left’. They correspond to some extent to ‘Present’ and ‘Dismissed’ for the candidates but also cover the fact that the location needs to be restored after the assessment.

### 2.3.2 Optional Alpha ‘System’

In case a computer-aided assessment system or similar electronic system is used to conduct the assessment, it can be represented by an additional optional alpha. The alpha covers all possible duties of this system such as administering the tests, accepting submissions, associating grades and feedback to submissions and performing grade and feedback generation automatically. This alpha is only relevant for electronic assessments. States and checkpoints for alpha ‘System’ are shown in Fig. 9.

Similar to the previous alpha, the names of the first two states are ‘Defined’ and ‘Selected’. This again reflects the fact that (at least in an ideal scenario) one would first define some abstract requirements towards the assessment system and then select and actual system fulfilling these requirements. In reality, organizers sometimes have no choice, as they must use the system provided by their institution. In this case, these two states are fulfilled by default and the features of the available system may restrict organizers in the selection of test item formats they can use. Since the ESSENCE notation does not require to define dependencies between states from different alphas explicitly, processes for both orders can be defined and monitored using this kernel. The name of the third state is ‘Available’. It refers to the fact that the selected system

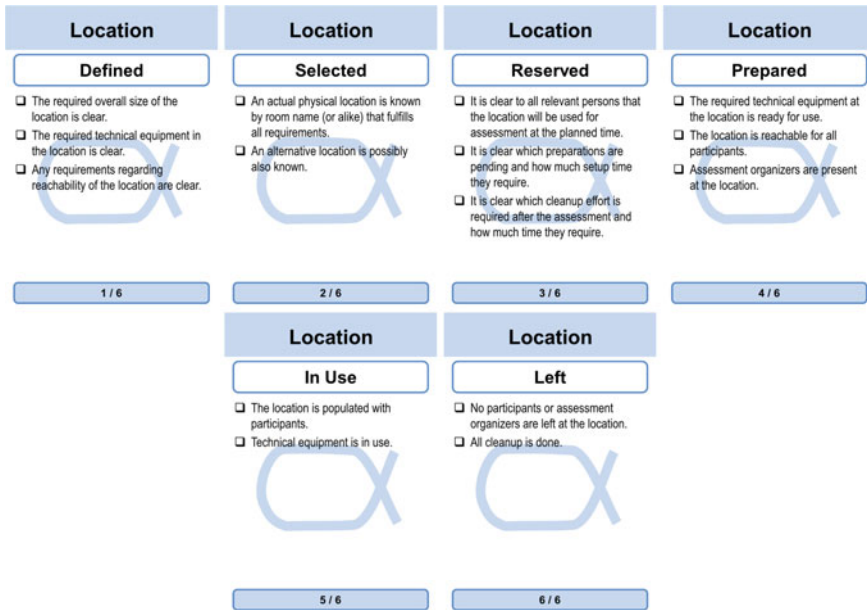


Fig. 8 Alpha states and checklists for the six states of alpha ‘Location’

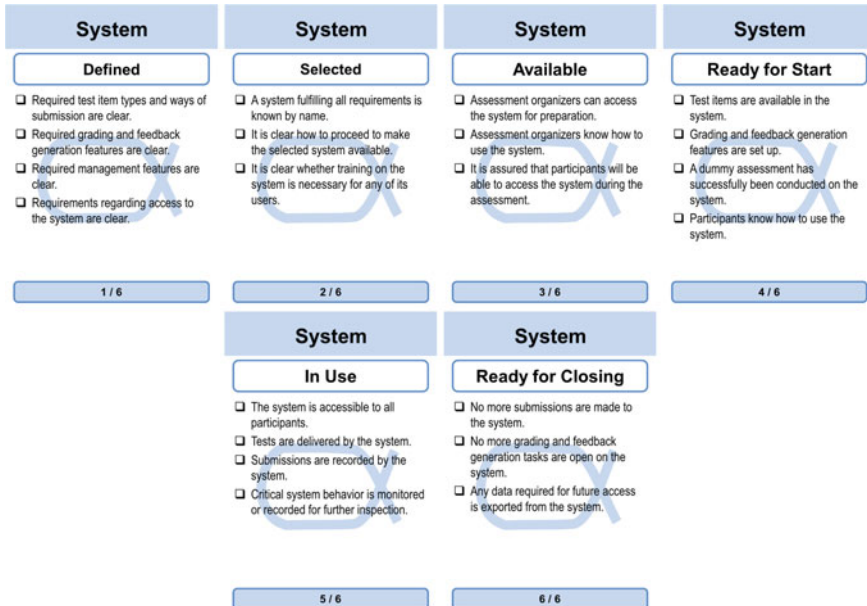


Fig. 9 Alpha states and checklists for the six states of alpha ‘System’

also needs to be accessible to continue preparation. This in turn will lead to the fourth state, which is named 'Ready for Start'. The name of the fifth state is 'In Use'. It depicts the period of time in which candidates interact with the system. This is also the period of time in which it performs tasks like automated grading on its own. The name of the final state is 'Ready for Closing'. This state makes no assumptions on whether the whole system will actually be closed or whether it is just the assessment that is closed and archived. However, it is assumed that any remaining steps of the process will not require any more interaction with the assessment system.

### 3 Sample Workflow Definitions

To demonstrate how to define processes based on the *Kernel for Educational Assessment* we consider a summative e-assessment such as an electronic exam. This scenario is based on practical experience of the author with exams held several times a year. In this scenario, candidates come to the exam hall that is equipped with computers and an appropriate e-assessment system. There is no need to provide direct feedback to the candidates while they are present in the exam hall so that solutions can be graded asynchronously. In fact, this scenario requires a quite large and complex workflow. However, it can be described in very lean and concise way, as the next sections will demonstrate.

The technique used to model the process is to group states from several alphas into a phase and define the process as a linear sequence of phases. One phase can cover more than one state of a single alpha, but there may also be alphas that do not contribute one of their states for a particular phase. There are other ways to model processes as well, e.g. linking alpha states via activities, but as neither activities nor activity spaces have been discussed in this chapter, this way of modelling is also ignored here. The process model thus comes close to what is suggested as 'big picture of assessment' as suggested in Banta and Palomba (2015).

Our scenario of a summative e-assessment can be described using five phases:

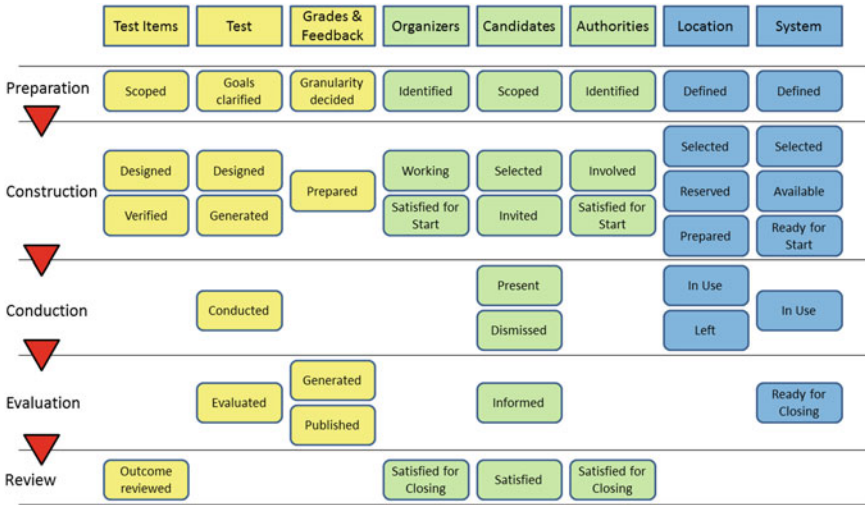
1. The *preparation phase* contains all states that are considered while planning the assessment. While scope, shape or the number of people involved in the assessment are not clear at the beginning of this phase, most of these bounds and circumstances should be made clear during this phase. However, states dealing with details that are considered of minor importance can be deferred to later phases. On the other hand, any state bearing major decisions about cancelling the assessment should be included into this phase, as cancelling later will result in wasting significant amounts of work.
2. The *construction phase* contains all states that relate in some way to the production of resources and artefacts needed during the assessment. It can be assumed that a significant amount of time will be spent on tasks arising from this phase. Any state that must be completed before that assessment starts should be placed in this phase at the latest.

3. The *conduction phase* represents the time frame in which the actual assessment is conducted. Thus, all states related to delivering tests, collecting submissions and monitoring the assessment should be grouped in this phase. In particular, this is most likely the only phase in which the candidates have direct contact with the assessment.
4. The *evaluation phase* bundles states related to assessing submissions or answers from the candidates and generating feedback. From the didactic point of view, this is one of the most important phases, as this phase produces the actual outcome of the assessment and thus contributes much to its overall value. Depending on the domain of the assessment, the test item types used and the mechanisms applied for grading, this phase can consume a lot of time in the whole assessment process. As we assumed asynchronous grading for our sample process, this phase is clearly distinct from the conduction phase.
5. The *review phase* is considered to be the final phase in the assessment process. It should cover both legal and organizational post-processing and also tasks on documenting how well the assessment process actually worked. It is likely that some people who have been involved in the assessment process so far have no duties in this phase and thus can leave the process early. Consequently, some alphas may have reached their final state already in an earlier phase and do thus not contribute to the review phase.

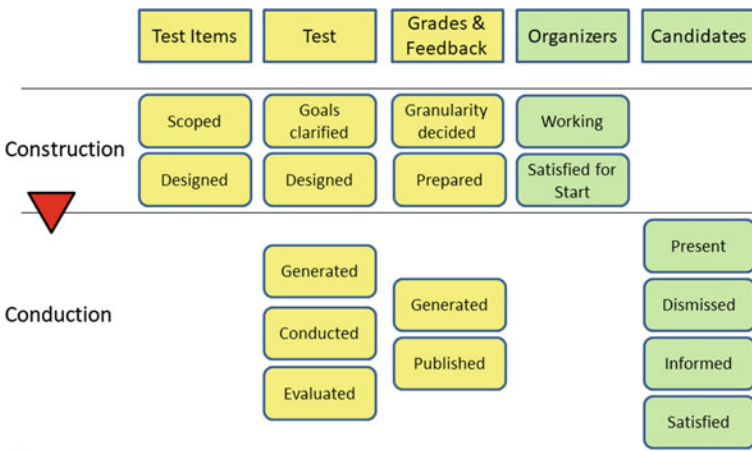
The resulting process description in terms of alpha states assigned to phases is depicted in Fig. 10. All alphas including the optional ones are used, as we employ an electronic system and have to involve the exam authorities. Notably, we can skip the alpha ‘System’ from the process and retain a process that represents a traditional written exam that is graded manually after conduction.

Although this is a concise representation of a complex process, the process itself is not very lean. However, the kernel and the phase model can also be used to represent much more lightweight processes by skipping not only optional alphas, but also some more alphas and also particular states of alphas. To illustrate this, we consider a second scenario in which an assessor interacts spontaneously with some candidates just where they are. This is what many educators do when running lab exercises or alike. In contrast to the scenario used before, we can expect to see a very lean workflow here. Consequently, it is quite unlikely that a process description for this scenario will be used to guide the assessor in this process, but it can be used descriptively to explain what is going on.

The process differs in several points from the one discussed so far: First, we can exclude alphas ‘Authorities’, ‘System’ and also ‘Location’, as the assessment is informal, includes no e-assessment system and can happen anywhere. Second, we can exclude several states of some of the involved alphas: As the assessor interacts with the candidates who are just present, we can exclude the first two stages of alpha ‘Candidates’. Thus ‘Present’ is the first state for candidates to be considered in this process. With similar arguments, we can also exclude state ‘Identified’ for alpha ‘Organizers’. Third, the scenario poses less strict requirements with respect to verification and review of assessment contents. Hence, we can exclude the last two



**Fig. 10** Overview on the assessment process for a summative e-assessment using five phases. The process assumes the application of asynchronous grading, so evaluation happens in a separate phase after conduction



**Fig. 11** Overview on the assessment process for a lightweight ad hoc assessment using just two phases. The very informal setting allows to skip the alphas ‘Authorities’ and ‘Location’ from the process description. Also ‘System’ can be skipped as this assessment is not considered to be an e-assessment

states for alpha ‘Test Items’ as well as the final state for ‘Organizers’. The resulting process description in terms of alpha states assigned to phases is depicted in Fig. 11.

The remaining states of the five alphas can then be grouped into just two phases. The construction phase consequently contains the first two states for ‘Test Items’, ‘Test’, ‘Grades & Feedback’ and ‘Organizers’. It thus describes the time frame in

which the organizer thinks about doing the assessment and plans what to ask. As we assume this scenario to be a spontaneous assessment, no preparations have happened before. Candidates are not involved in this phase. The other phase is the conduction phase in which only ‘Test’, ‘Grades & Feedback’ and ‘Candidates’ are involved in terms of changing states. This phase is rather similar to the one seen above besides the fact that state ‘Satisfied’ for alpha ‘Candidates’ is also included here. The idea is that in an ad hoc assessment, any appeals are handled directly and thus no formal review phase is needed. As already discussed above, the organizer is also not interested in detailed verification and review. Thus, the respective states from the review phases in the other case studies are simply skipped here.

One could think of making the process description even smaller by skipping state ‘Dismissed’ for alpha ‘Candidates’. This would stress the point that the assessment can happen anywhere and candidates are not required to come to a certain location (and consequently leave it later). On the other hand, one can understand the state ‘Dismissed’ also in a less literal way and consider a candidate dismissed once the organizer stopped asking questions to this candidate. Notably, the ESSENCE standard allows to make customizations to states in terms of adding or removing checklist items. Consequently, one could define an even more fine-grained adoption of the kernel for this particular scenario by changing the checklists but keeping the overall idea of each of the alpha states included in the process description.

## 4 Tool Support

The ESSENCE standard defines the notion of cards for each alpha state. This idea was also used throughout this chapter to present the different alphas. Hence, a very native way of tool support is to print out small cards and use them on a pin board or desk to arrange them into phases and tick off checklist items. However, this is hardly practical for educators who have to prepare and run several different assessments in parallel. Electronic tool support can be considered much more practical in this case. On the other hand, tool support in terms of strict workflow engines is less appropriate when agile processes are to be used.

A simple web application that provides an overview on a process but deliberately provides no automated enactment of processes is shown in Fig. 12. It is based on an industry tool for software engineering process management (Brandt, Striewe, Beck, & Goedicke, 2017) that can be used with different kernel and process descriptions based on the ESSENCE standard.

There are two ways how educators can use this tool to handle their workflows: First, they can use it to provide a concise description of their assessment process. This can be helpful when discussing processes with colleagues or comparing different assessment workflows. At the same time, they can use it to customize their workflows by moving states to different phases, hiding alphas or adding specific checklist items. As the tool does not provide any formal workflow engine as other tools do, it also does not make any constraints on how educators can change a process description.

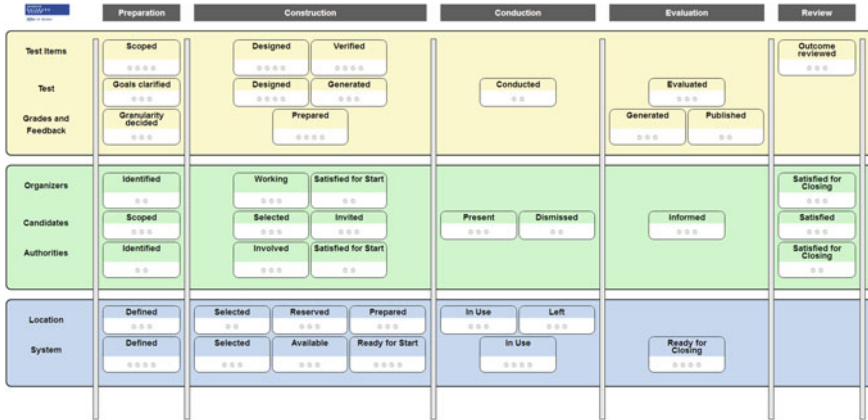


Fig. 12 Web application showing an overview on the sample process from Fig. 10. Phases run from left to right here instead of top to bottom. Users can click on the small boxes to get a detail view for each alpha state and to tick off checklist items

Second, educators can use the tool to enact and monitor their workflow by ticking off checklist items and thus tracking progress. The tool indicates fulfilled checklist items and states with different colouring, so educators always can see what is already done and what has to be done next. Again, the tool does not force them to perform a particular activity at a particular point in time, but allows an agile enactment in which the educators set their own goals. The tool also allows working collaboratively and thus sharing the responsibility for a particular assessment process. Practical experience with staff responsible for managing assessment processes in universities revealed that they prefer using a tool like this with customized workflows over using general workflow management tools. They also preferred using a tool like this over managing assessment processes by hand or with standard office tools.

## 5 Summary and Discussion

This chapter introduced the *Kernel for Educational Assessment* and demonstrated how to model lean and agile assessment workflows. The kernel contains a universal set of elements that can be used as building blocks for individual workflow descriptions. Each element is small and has a concise representation. Practical experience as expressed in the two sample processes shows that there are enough elements for complex workflows. At the same time, the set can also be stripped down to a very small number of elements used in very lightweight assessment processes. The notion of states and checkpoints can be used both for describing an assessment process and for monitoring the workflow while enacting it.



Notably, the process of stripping down a complex workflow by removing alpha states looks very mechanic. This seems to be an interesting antithesis to the ideas of agile processes on the first glance. However, one has to watch the different meta-levels: The way of describing processes is somewhat mechanic. The processes themselves can be as complex or lean as needed and can be changed in an agile way whenever needed.

## References

- Banta, T. W., & Palomba, C. A. (2015). *Assessment essentials* (2nd ed.). Wiley.
- Brandt, S., Striewe, M., Beck, F., & Goedicke, M. (2017). A dashboard for visualizing software engineering processes based on ESSENCE. In *5th IEEE Working Conference on Software Visualization (VISSOFT)*.
- Danson, M., Dawson, B., & Baseley, T. (2001). Large scale implementation of question mark perception (V2.5)—Experiences at Loughborough University. In *Proceedings of the 5th Computer-Assisted Assessment Conference (CAA)*.
- Dick, W., Carey, L., & Carey, J. O. (2014). *The systematic design of instruction* (8th ed.). Pearson Education.
- Essence—Kernel and Language for Software Engineering Methods*. (2015, Dec). Retrieved from <http://www.omg.org/spec/Essence/1.1>.
- Jacobson, I., Spence, I., & Ng, P.-W. (2013, September). Agile and SEMAT—Perfect Partners. *ACM Queue*, 11(9), 30:30–30:41. <https://doi.org/10.1145/2538031.2541674>.
- Johnson, D. H., & Johnson, R. T. (2002). *Meaningful assessment: A manageable and cooperative process*. Pearson.
- Laird, D., Holton, E. F., & Naquin, S. S. (2003). *Approaches to training and development*. Basic Books.
- Millard, D. E., Bailey, C., Davis, H. C., Gilbert, L., Howard, Y., & Wills, G. (2006, July). The e-learning assessment landscape. In *Sixth IEEE International Conference on Advanced Learning Technologies (ICALT'06)* (pp. 964–966). <https://doi.org/10.1109/icalt.2006.1652604>.
- Reynolds, C., Livingston, R., & Willson, V. (2009). *Measurement and assessment in education*. Pearson.
- Wills, G. B., Bailey, C. P., Davis, H. C., Gilbert, L., Howard, Y., Jeyes, S., ... Young, R. (2007). An e-learning framework for assessment (FREMA). In *Proceedings of the 11th Computer-Assisted Assessment Conference (CAA)*.

**Part IV**  
**Agile and Lean Learning Processes**

# Criterion-Based Grading, Agile Goal Setting, and Course (Un)Completion Strategies



Petri Ihantola , Essi Isohanni, Pietari Heino and Tommi Mikkonen

**Abstract** When teaching large groups of students with heterogeneous backgrounds and different learning goals, it is essential to personalize the learning experience. In this chapter, we describe how we have implemented this in a university-wide introductory programming course. Each student sets a personal target grade, i.e., the grade they aim at, based on how deep an understanding of programming they need (depending on their major subject, etc.) and on how much effort they are willing to invest in the course. To enable such setup, course assignments are divided into different levels and the grading directs the students in choosing which assignments to work on to meet the goals they have set. Furthermore, the students can change their target grade during the course in an agile manner.

**Keywords** Criterion-based grading · Automated assessment · CS1 Student strategies · Agile goal setting

## 1 Introduction

The constructivist learning theories propose that a learner constructs their own comprehension of the subject through their prior knowledge (Illeris, 2002). These theories emphasize that learning is an individual process that reflects the background of the

---

P. Ihantola (✉) · T. Mikkonen  
University of Helsinki, Helsinki, Finland  
e-mail: [petri.ihantola@helsinki.fi](mailto:petri.ihantola@helsinki.fi)

T. Mikkonen  
e-mail: [tommi.mikkonen@helsinki.fi](mailto:tommi.mikkonen@helsinki.fi)

E. Isohanni · P. Heino  
Tampere University of Technology, Tampere, Finland  
e-mail: [essi.isohanni@tut.fi](mailto:essi.isohanni@tut.fi)

P. Heino  
e-mail: [pietari.heino@tut.fi](mailto:pietari.heino@tut.fi)

learner. Therefore, it is essential to let the learner personalize the learning process by choosing learning materials according to personal preference.

In self-directed learning—typical in adult education—the learner also takes the initiative to formulate and pursue learning goals (Merriam, 2001). Unfortunately, this self-imposed setting of goals is often poorly supported. As Lister and Leaney (2003) state:

In the traditional [...] approach to grading, all students in a CS1 class attempt the same programming tasks, and those attempts are graded “to a curve”. The danger is that such tasks are aimed at a hypothetical average student. Weaker students can do little of these tasks, and learn little. Meanwhile, these tasks do not stretch the stronger students, so they too are denied an opportunity to learn.

In the same article, Lister and Leaney propose a criterion-referenced grading scheme, where the students do different assignments, according to their abilities. Moreover, the assignments are designed to match the cognitive domains of Bloom’s taxonomy (Bloom et al., 1956), a classification of levels of intellectual behavior important in learning (Seddon, 1978). While the taxonomy consists of three hierarchical models—cognitive, affective, and sensory domains—the cognitive part has been the primary focus of most traditional education. In particular, it has been commonly used as the basis for structuring curriculum learning goals, assessments, and activities (Fuller et al., 2007).

In this chapter, we describe how we have implemented the criterion-referenced grading scheme in the context of a university-wide introductory programming course. In addition, we introduce an agile course concept: We explicitly ask each student to choose their learning goals, which define the assignments they should complete. In connection to agile software development, the analogy is to allow the team to decide which features to pick from the product backlog. Here, the students decide how much work they are willing to invest in learning a certain topic in the course, and pick assignments with corresponding complexity. The approach was designed to support both struggling (Ahadi, Lister, Haapala, & Vihavainen, 2015) and overperforming (Carter et al., 2010, 2011) students by providing a personal, agile learning experience despite extremely large teaching groups.

Struggling and over-performance are often related to a mismatch between prior skills and learning goals. We have divided all our programming assignments into four categories, with increasing complexity. The way this is done resembles grouping the levels of Bloom (Lister & Leaney, 2003; Johnson, Gaspar, Boyer, Bennett, & Armitage, 2012). However, as applying Bloom’s taxonomy consistently in Computer Science (CS) education can be very challenging (Johnson & Fuller, 2006; Thompson, Luxton-Reilly, Whalley, Hu, & Robbins, 2008), our approach is more practically oriented: Skills learned from the higher category assignments are prerequisites of a future course only if a student is planning to take programming as their major or minor. Moreover, assignments from the lower categories may be too simple for students with prior knowledge. Thus, they are optional for students aiming at higher grades.

Our main tool for organizing the course in an agile way is the grading rules of the course. In our setup, the final grade is based on solving automatically graded

programming assignments throughout the course. The approach is applicable also in other contexts, however. We describe the course to give an overall understanding and to reflect our results. In addition to explaining the details of how we have implemented agile into education, our main objective is to seek understanding regarding how students behave in the setup.

The rest of the chapter is structured as follows. First, in Sect. 2, we introduce the agile course setting, which forms the background of this paper. Then, in Sect. 3, we describe the research questions related to students' behavior and the related methodology. Next, in Sect. 4, we introduce our results, and in Sect. 5, we provide an extended discussion regarding our main observations. Finally, in Sect. 6 we draw some conclusions.

## 2 Agile Course Setting

In order to describe how the agile goal setting has been implemented in our introductory programming course, we first describe why the course is needed in our university and who takes it (Sect. 2.1). Next, we define the teaching methods (Sect. 2.2), and the grading scheme enabling the individual learning paths (Sect. 2.3). Finally, the idea of the grading scheme is illustrated with examples of different learning goals (Sect. 2.4).

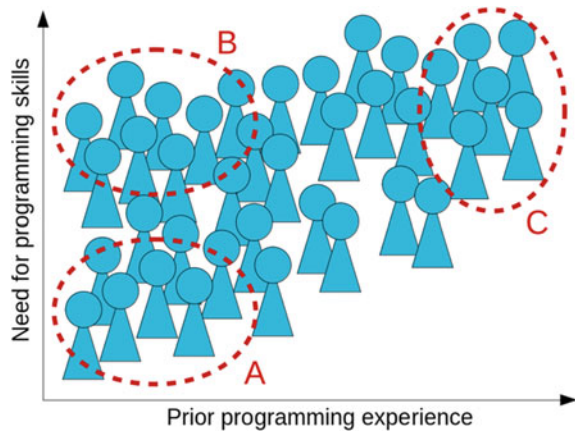
### 2.1 *Versatile Needs*

The overarching learning goal of the Introduction to Programming course in Tampere University of Technology, Finland, is to learn to write small programs on one's own. We use Python as the programming language, and in addition to basic computer usage skills, there are no other prerequisites for the course.

The course is obligatory for almost all the students in the whole university, more than 1000 students every year. Consequently, large and heterogeneous student groups are included—in addition to computer science students, also, for instance, electrical engineering, automation engineering, mechanical engineering, and even material science students take the course. The background for this decision is that in the modern world all graduates need to know at least the basics of programming to better understand the use of computer applications in their own fields.

While computer science students build almost all of their professional skills on top of their ability to program, the students in many other fields just need to understand what programming is. In addition, students' previous programming skills vary greatly: most of the students start with no previous programming knowledge, but the teaching group also includes students who have been programming in high school and in their free time.

**Fig. 1** Diverse student population, with special target groups A, B, and C identified



The diversity of the teaching group is illustrated in Fig. 1. Students in Group A are majoring in some other field than computer science. Consequently, they only need elementary programming skills. Group B consists of students who start with no prior programming experience and need to cover all the topics of the course. They need to work really hard in this course. Group C consists of students that know the basics prior to the course. Thus, they cope even without special attention from the teacher and are most often neglected in large teaching groups. In our case, all of these students are attending the same course, and hence our goal is to meet the needs of all these student groups in our pedagogical design.

Another solution would be that all the different curricula that need programming would be free to create their own courses with small number of students participating in each of them. However, our solution where all students attend the same course ensures flexible possibilities of changing study plans for students. There are many students who do not know what programming is before they attend the course. In our course, they can decide that they want to cover the topics more thoroughly and proceed to further programming courses. In addition, in comparison to going for different courses, this model allows fine-tuning the goals in accordance to students' views, not only on individual course definitions.

## 2.2 Practical Arrangements

All the course material is available online, delivered by using the A+ course platform (Karavirta, Ihantola, & Koskinen, 2013). The material contains an e-book where all the theory is explained and embeds automatically assessed programming assignments with immediate feedback. Each assignment can be submitted for evaluation at most ten times. This kind of automated approach is widely applied in programming education (Carter et al., 2003; Douce, Livingstone, & Orwell, 2005).

In addition to automated feedback, the students also receive feedback from teaching assistants for selected assignments. These selected assignments test the core ideas of programming thus giving the students first-hand feedback on whether they have actually understood the problems at hand or if there is something to improve on and pay attention to in the future.

The course spans over two teaching periods (in total 14 weeks), and it is five ECTS credits in size. A number of assignments are to be completed weekly. We use the flipped classroom method (Bishop & Verleger, 2013), where the students first complete the programming assignments and then attend the class to discuss different ways of solving the assignments and the problems they faced. Students work self-guided and ask for help from the teaching staff if necessary. To this end, we use a multipurpose computer classroom, where the students are allowed to work whenever they want. In addition, there are teaching assistant hours in the multipurpose classroom at least 20 h a week.

At the end of the course, students take an electronic exam which is like a skills demonstration where the student shows under controlled conditions that he/she can complete a small programming assignment independently. The exam is not supposed to be difficult. Essentially it is a programming assignment of the same style as provided during the course.

Our course design is learner-centered. The students work according to their personal weekly schedules and the teaching assistants are available upon request in the multipurpose classroom at least during peak times. This way of working requires a lot of self-discipline from our students. On the other hand, it is very flexible. We also highlight that working in their own schedule does not mean working alone. They are encouraged to work in pairs and also to discuss their problems both with the teaching assistants and in the lectures.

The course design and all the practical arrangements follow the principle of constructive alignment (Biggs & Tang, 2007) in which teaching and assessment methods are chosen to meet predetermined learning goals. As the learning goal is to learn to implement small programs on one's own, that is exactly what the student does every week during the course in all the assignments and in the exam.

### 2.3 Grading

Our course is graded on a scale from zero (failed) to five (the best). The final grade is defined by the completed assignments. In the A+ course platform, there are over 120 programming assignments, which we divided into four categories:

- **A elementary**: small assignments, where the student mostly just repeats something that was exemplified in the materials.
- **B basic**: further small assignments, which mainly concentrate on one new topic, but are more difficult than elementary assignments because the solution is not directly in the materials.

**Table 1** The grading rules table lists what are the requirements for each grade

Grade	A-points (elementary)	B-points (basic)	C-points (applied)	D-points (advanced)	Exam2
1	600	700	–	–	Basic
2	600	800	200	–	Basic
3	400	900	400	–	Extended
4	200	900	400	200	Extended
5	–	700	400	500	Extended

- **C applied:** typically, the student has to combine knowledge related to more than one topic. Some of these assignments are so large that we call them projects, despite the students working on them only for a few hours.
- **D advanced:** these assignments require the student to get familiar with materials outside the core content of the course or are in some other way more difficult than assignments in the other categories.

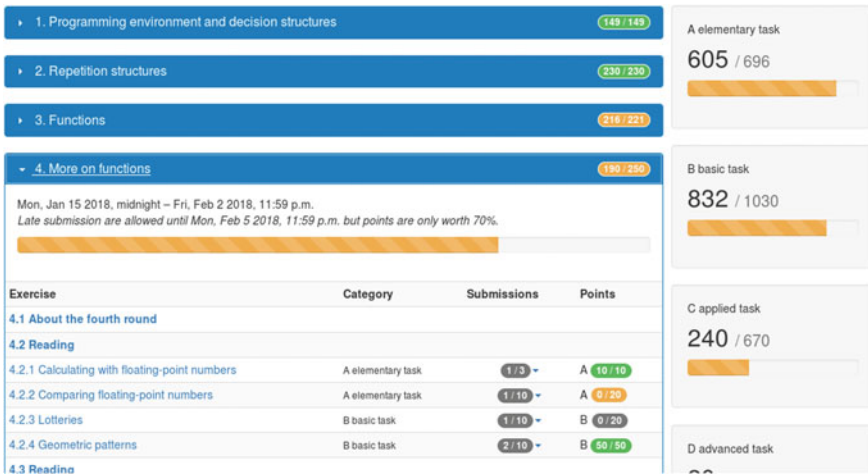
Points are allocated to assignments in accordance to how laborious they are. The largest ones constitute approximately 100 points and the smallest 10 points. There is a deadline every week, with an optional, discouraged extension. If an assignment is submitted during the extension, one can only receive 70% of the associated points. Even during the extension, it is possible to meet the teaching assistants in the class. This is important, because immediate automated feedback may reveal misunderstood requirements and other problems.

Assignments are associated with grading rules. Table 1 presents a slightly simplified version of them. To achieve grade 3, the student should collect 400 points from the elementary, 900 points from the basic, and 400 points from the applied assignments, and pass the extended exam. Both exams are only graded as pass/fail. The verbal explanation for grade 3 is “good” in our university. Thus, the requirement is that the student achieving this grade knows the core content of the course well enough to apply it in practice (applied assignments, level C). Students targeting grades higher than 3 also need to accomplish advanced assignments. If the student has set the target lower than grade 3, he/she is allowed to pass the course with less work. However, a grade of 3 is a prerequisite for the more advanced programming courses.

As different students achieve different skill levels, there are two versions of the exam, basic and extended. The grading rules (Table 1) also define which exam the student has to take. The extended exam is not necessarily any harder than the basic one, but it covers a wider selection of topics. The role of the exam is only to double-check that students did their assignments independently enough to reach the learning objectives. Therefore, the exam is graded pass/fail and failing it will cause failing the course.

It is possible to patch up missing points by completing assignments from the higher categories. The purpose of this rule is to enforce that missing a single deadline is never fatal. If you miss deadlines, the first consequence is that it is impossible to gain





**Fig. 2** Screen shot of the course platform showing progress bars and completion status of different assignments

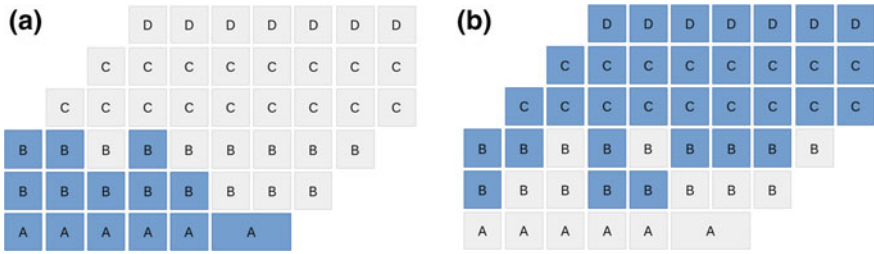
the grade 5. By missing more deadlines, the grade 4 will also be unreachable, and so on. To fail the course completely, the state of affairs has to be such that the student has repeatedly missed deadlines.

On the course platform, the student is presented with progress bars showing the number of collected points in each category. It is easy to follow how the points accumulate. In addition, the fully completed assignment scores are shown in green-colored circles and partly completed in yellow. Figure 2 shows the students view of the course platform. On the right-hand side, the student sees the accumulated points from different assignment categories and in the middle the progress of different rounds; the first four rounds (weeks) that have already been closed are collapsed while the last one is open.

### 2.4 Individual Learning Goals and Learning Paths

As already pointed out, the overarching learning goal of the course is that the student is able to implement small programs independently. However, in the spirit of the criterion-referenced grading (Lister & Leaney, 2003), with such a diverse student group in question, the students' individual learning goals may vary greatly.

All the students passing the course are required to meet the learning goal of implementing small programs independently. This means all students cover the basic concepts of Python—i.e., functions, lists, dicts, for instance—in some way. In levels A and B, the learning goal is just to get familiar with all these concepts. In level C, the students need to be able to apply this knowledge in more difficult situations.



**Fig. 3** Illustrating the differentiation of learning paths in our course setting

For example, in addition to knowing the data structures list and dict, they need to be able to combine the data structures, i.e., implement a program where the items of a list are dicts, for instance. In level D, the assumption is that the student will go further with programming studies and thus needs to learn how programmers search for information in work life. Therefore, some of the D assignments require searching for information in the Python documentation.

In contrast to traditional grading, here the grade that the student receives does not describe how well the student can implement given tasks but how widely he/she has covered the topics handled in the course.

Setting individual learning goals does not mean that the students who only cover assignments in levels A and B cover all the materials but do it somehow worse than other students. It means that they have identified they do not need to cover all the materials. They can still do all their work thoroughly. For example, for a student of some discipline other than computing, it is enough that he/she is able to apply the information provided in the course materials and there is no need to learn to use the Python documentation.

Figure 3 presents two simplified illustrations regarding how the course can proceed. The time-axis of the illustrations runs horizontally from left to right. There are a number of assignments on different levels each week. In the beginning of the course, there are mainly elementary and basic assignments and, in the end, applied, and advanced. Figure 3a illustrates the course completion of a student who has set a very low target grade and only worked on level A- and B-assignments. Figure 3b illustrates the course completion of a student who has had prior programming experience already when starting the course and thus has skipped all the A-level assignments. When comparing these two accomplishments, we can almost say that these two students have almost taken a different course because their learning paths differ so much from each other despite them taking the same course.

This way of implementing differentiation of learning paths allows us to pay special attention to the student group C (Fig. 1), which is often neglected simply because other student groups need more attention. As no student in the course is supposed to complete all assignments available on the course platform, some of the advanced ones can be so difficult that they also challenge the students with previous programming

experience. Some of the advanced assignments can also cover topics that are not important for all the students.

The students—especially those with no prior programming experience—are encouraged to complete as many assignments as possible in the first weeks of the course. If they do not know anything about the course content, it is almost impossible to set the targets. At the end of the third week of the course, they have already seen assignments at all difficulty levels and know a little better so it is possible to set the target grades. At this point, we inquire what their targets are for this course. Despite their answers, nothing prevents the students from changing their targets later on, however. For example, many of the advanced assignments are “deep diving” into a specific topic. Skipping advanced assignments in the first half of the course does not prevent the student from completing assignments from the advanced category in the latter half of the course. Naturally, even easier is to stop working on the assignments from the higher levels.

### **3 Research Questions and Methodology**

In the previous section, we presented the agile grading scheme and how we assumed students would apply it. In this section, we describe the research setup aiming at understanding how students really use the setup.

#### ***3.1 Research Questions***

In addition to describing our implementation of the agile course setup, the main objective of this study is to understand how students utilize the agile course setup defined in the previous section. The related research questions are as follows:

1. How are students’ self-reported target grades (i.e., individual learning goals) in the beginning of the course related to the final learning outcomes?
2. What kinds of behavioral patterns can be detected among students in the agile course setup?

Studying these research questions helps us to understand the course setup from the perspective of the diverse student population (Fig. 1) attending our course.

#### ***3.2 Data***

To answer the research questions, we collected data in two course implementations during the academic year 2016–2017. The CS students were mainly attending the first implementation (Autumn 2016). Students from the other disciplines attended

both implementations (Autumn 2016 and Spring 2017). We had access to log data of all the submissions from the course platform, grade targets set after the third week and final grades.

When comparing the grades in RQ1, we analyzed the data of all the students who were active in the course platform during any 2 weeks of the course, or who answered the question about the grade target. This is because, in our context, it is quite common that students sign up to the course platform just to see what the course looks like and may then immediately drop out from the course. The selected inclusion criteria resulted in 831 students; 527 in the 2016 course version from where 482 (91%) answered the grade target question, and 304 in the 2016 course version from where 267 (88%) answered the question. When answering the RQ2, we focused on the subset of the students who had answered the question about the grade target.

### 3.3 Methods

To answer RQ1, addressing the effect of defining an individual learning goal, we used Wilcoxon signed-rank test to compare the grade distributions of students who set their grade targets and students who did not. In addition, for students who set a target, we examined the predictive power of a linear regression model to estimate the final grade based on the target.

In RQ2, we applied visual data analytics to identify different behavioral patterns among the students taking the course. As defined by Keim (2002): “*The visual data exploration process can be seen a hypothesis generation process: The visualizations of the data allow the user to gain insight into the data and come up with new hypotheses*”. This kind of manual exploration of visualizations is typical in learning analytics and educational data mining (Mazza & Milani, 2005; Romero & Ventura, 2010).

Two teachers of the course who had knowledge on the content of the assignments looked at the visualizations and identified students with similar characteristics in their course completion paths. Examples of visualizations will be provided later on in this paper in Sect. 4.2. Patterns were identified and the data was then reanalyzed in order to confirm visualizations that can be divided into these categories. Although visualizations of all study paths were viewed for identifying archetypal course completion patterns, the exact frequencies of the different learning strategies were not calculated.

## 4 Results

In this section, we describe the phenomena discovered in the data using both the visualizations and the additional data related to the course context. Section 4.1 is related to RQ1 and Sect. 4.2 to RQ2.

### 4.1 Comparison of Grades and Targets

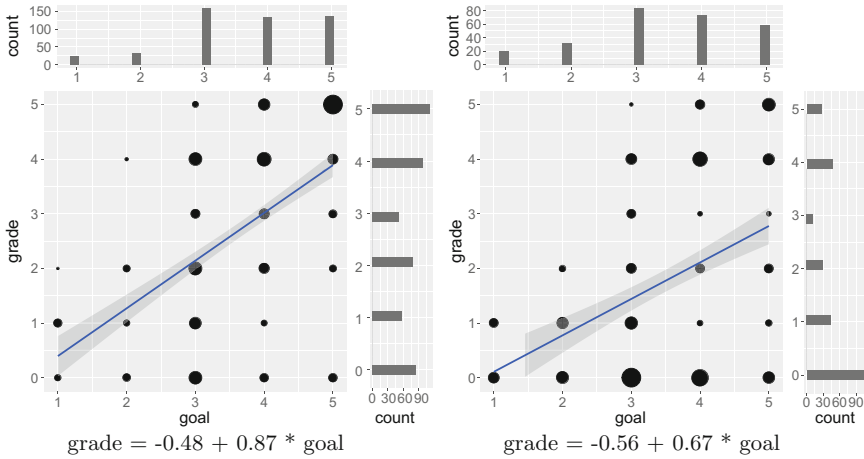
Table 2 presents the distributions of the final grades in both course versions. Presentation separates students with and without grade target set. As expected, grades in the autumn 2016 course version (with CS-majors) are higher than in spring 2017 when participants were mostly nonmajors.

We calculated the Wilcoxon signed-rank test with continuity correction to compare grades of students who answered the questions about the goal grade and students who did not. Nonparametric method was selected because of the relatively small number and skewed distribution of the students who did not set their target grades (see Table 2). Students who answered the question performed significantly better with  $Z = 5232, p < 0.000$  and  $Z = 3683, p = 0.004$ , respectively, in 2016 and 2017 course versions. We conclude that merely setting a grade target (i.e., an individual learning goal) seems to play an important role in performance. The grade difference in terms of median grades in 2016 was one and in 2017 two grades. Defining a goal seems to be especially important in passing the course. As illustrated in Table 2, dropout rates in 2016 were 17.4% and 42.2%, respectively, for students with and without an individual goal. The same stats in 2017 were 40.1 and 70.3%.

For students who set their targets, we constructed simple linear regression models to predict the final grade based on the self-reported grade target. Both course versions resulted to a significant model with adjusted  $R^2$  of 0.277 ( $F(1,481) = 185.71, p < 0.000$ ) and 0.18 ( $F(1,265) = 58.89, p < 0.000$ ), respectively, for the 2016 and 2017 course versions. The exact models and related illustrations are provided in Fig. 4. Negative offset and slope less than 1 in both models indicate that estimates are often optimistic. Means of the target grades among the students who defined that were 3.7 and 3.4, whereas means of the actual grades were 2.7 and 1.7, respectively, for the 2016 and 2017 course versions. Moreover, we found that it is almost impossible to exceed your own expectations if the grade target is low (i.e., 1 or 2). Only one student who set the target grade at 1 or 2 achieved a better grade. Among the students who set a reasonable target grade, e.g., 3 or 4, there were more of those who exceeded their expectations (see discussion on casual well-performers in Sect. 4.2).

**Table 2** The grade distribution of students in both the course implementations

Grade	2016		2017	
	Goal set	No goal	Goal set	No goal
0	84 (17.4%)	19 (42.2%)	107 (40.1%)	26 (70.3%)
1	58 (12.0%)	13 (28.9%)	44 (16.5%)	4 (10.8%)
2	79 (16.4%)	7 (15.6%)	30 (11.2%)	–
3	51 (10.6%)	5 (11.1%)	12 (4.5%)	1 (2.7%)
4	99 (20.5%)	–	47 (17.6%)	4 (10.8%)
5	111 (23.0)	1 (2.2)	27 (10.15)	5 (5.4%)



**Fig. 4** Self-reported grade goal against the final grade and the regression model to predict the latter based on the target grade. Autumn 2016 course version (with CS students) is on the left. Spring 2017 course version with only a few CS students attending is on the right. The difference in student background is clearly visible in grade goals as well as eventual grades

### 4.2 Students' Strategies

To answer RQ2 (what kinds of behavioral patterns can be detected among students in the agile course setup) we started by looking at students who got the best grades. The rationale of this was to identify different strategies that lead to similar outcomes. After manual inspection of the data, we identified the following archetypes of students:

1. **Perfectionists**, who set their goal high and do practically all the available assignments.
2. **Opportunists**, who set their goal high, and did not do the optional (i.e., easier) assignments.
3. **Casual well-performers**, who did not set their goals too high but who still got a good grade at the end.

The exact frequencies of the patterns are not calculated. However, in comparison to perfectionists, the number of opportunists is small. Among casual well-performers, there is a continuous spectrum from perfectionists to opportunists, again dominated by perfectionists—or at least students who do more voluntary A-exercises. This is not surprising, as casual well-performers started with the assumption they will actually need some A-points (i.e., lower grade target). In addition to high performers, dropouts turned out as an interesting subgroup and the following patterns were recognized among them:

4. **Dropouts by missing assignments**: Students who after some point were not able to pass the course or get the desired grade and stopped immediately after that.

5. **Dropouts by missing skills:** Students who completed enough assignments and even participated in the exam (multiple times) without passing it.
6. **Dropouts by failing personal expectations:** Students who gained enough points to pass the course but not enough to meet the requirements of the target grade failed the course because they did not show up in the exam.

In addition to the above observations, two generic behavioral patterns that extend over the whole student population, almost disregarding their eventual grades, are as follows:

7. **Goal-driven students** will stop all their activities immediately after they have gained enough points for their target grade.
8. **Compensating students** who have missed a significant number of assignments but who then end up compensating the missing points by doing more demanding assignments.

In the following, we study these groups in more detail by using visualizations of their course completion paths.

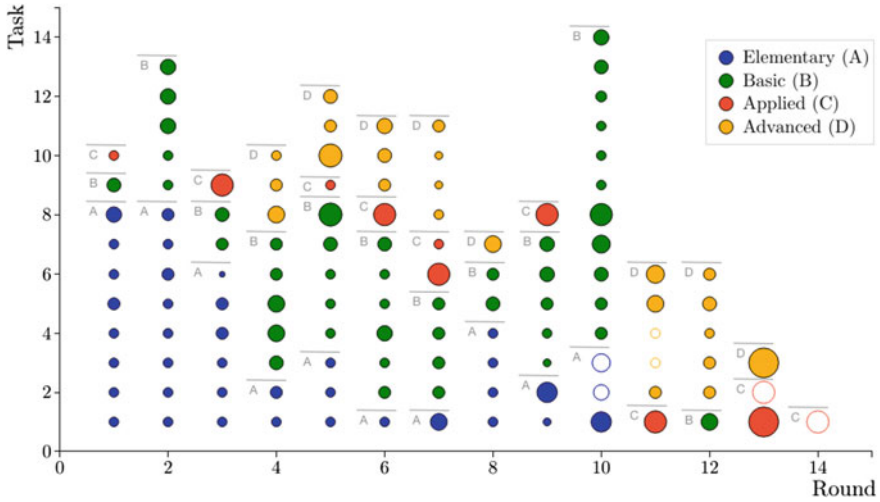
**Group 1: Perfectionists:** This group of students did not care for the point requirements much. They completed all or almost all the assignments despite they had already gained enough of points for grade 5. There were more of these students in the first-course implementation that the CS students took.

Figure 5 illustrates a typical course completion path for a student in this group. It is typical for enthusiastic learners to be interested in completing challenging assignments like the ones we had in level D. However, this student has also completed almost all A-assignments. It is especially noteworthy, that the A- and B-assignments in week 9 were all clearly instructed to be aimed for students who did not cover this topic in the earlier week. The students who had covered it, for instance, the student presented in Fig. 5, were instructed to proceed directly to the C-assignment of this week. Still, this student has completed them all, exactly like all the other assignments too.

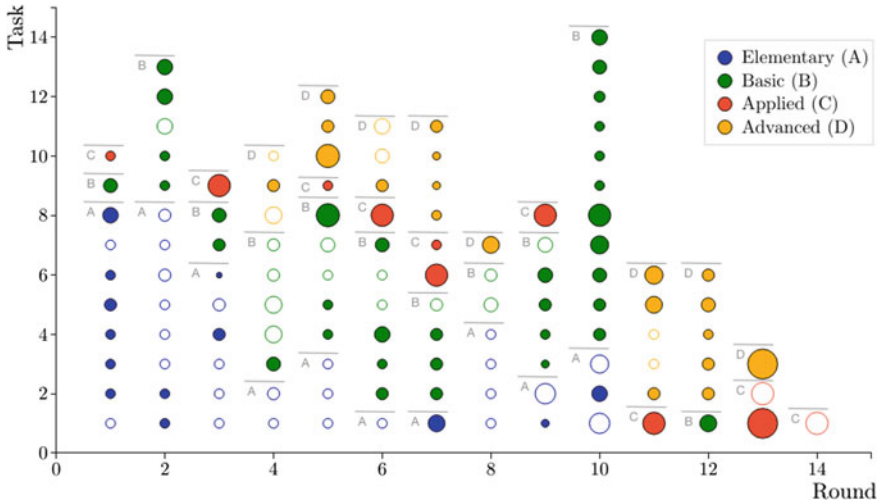
For this student, the course platform contains lines of green-colored circles and full progress bars. We assume that this was one of the motivational reasons for completing so many assignments that did not contribute to learning. This is why we have named the group perfectionist

**Group 2: Opportunists:** Although it was not usual, there were also students who took the advantage of not having to complete the A-level assignments since they were targeting grade 5 and had prior programming experience. Figure 6 illustrates one such completion path. In some of the weeks, this student has completed some of the easiest assignments probably to check what the new materials are about. However, he has mainly concentrated on completing the applied and advanced assignments, which is more meaningful for a student who already has the basic knowledge. The week 10, where he has completed multiple B-assignments, was about object-oriented programming, which was probably a new topic for this student.

**Group 3: Casual well-performers:** As can be perceived in Fig. 4, a number of students targeting grades 3 and 4 exceeded their personal targets and ended up with a better grade. The course completion path visualizations of these students do not



**Fig. 5** Perfectionist. Student who targeted grade 5 and achieved it. He completed practically all the assignments in the whole course. His only non-completed assignments are the so-called “extra assignments” that were open after the end of the course and assignments he was not required to do. Colors of the circles represent the assignment level and are explained in the upper right corner of the figure. Empty circles illustrate unfinished assignments. If reading the article in black-and-white, there are horizontal lines with equal labels between all the color groups



**Fig. 6** Opportunist. Student who targeted grade 5 and achieved it. He had previous programming experience before this course and thus took advantage of the grading rules. He has only checked some of the A-level assignments but mainly skipped them



differ much from the visualizations of groups 1 and 2. We identified these students by also looking at the targets they had set for themselves in the third week of the course.

These students probably did not know exactly what to expect from this course. They just started completing the assignments and kept on working even if they had reached the point requirements of the grade they were targeting. The aim of the grading system was that you can change your targets during the course and this group of students proves that it works.

**Group 4: Dropouts by missing assignments:** Some students missed so many assignments that it was not possible to compensate anymore. They gave up with the course and stopped working on the programming assignments. Typically, these students collected more points in the beginning of the course but then suddenly changed the direction completely.

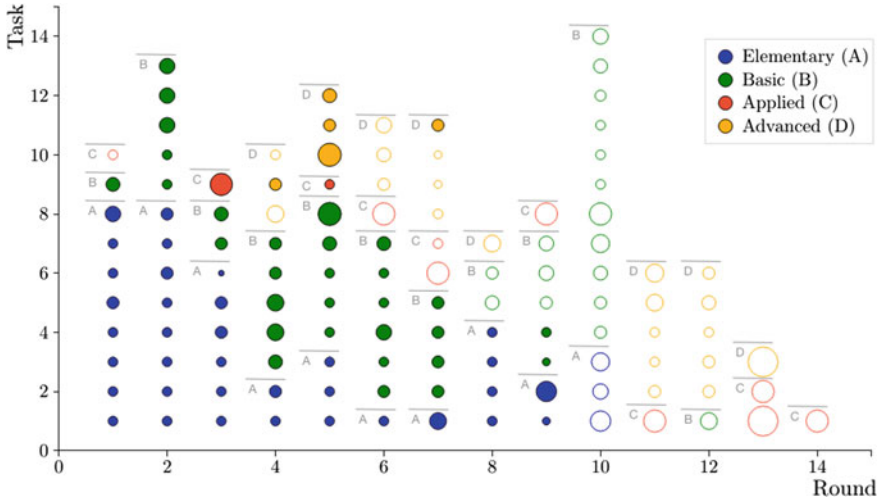
There can be various reasons for this behavior. Some of the students realized that completing the course required more time than they had expected and allocated in their weekly schedules. Some of the students realized that they have already missed so many points that it was impossible to gain enough points by completing all the remaining assignments.

**Group 5: Dropouts by missing skills:** These students completed a lot of programming assignments and gathered enough points to pass the course. However, they failed due to not passing the exam. As explained earlier, the exam was not a difficult programming assignment, but similar to the regular assignments done during the course. What made the exam different from the regular assignments was that the student was expected to demonstrate the ability to write some code on their own in a controlled environment.

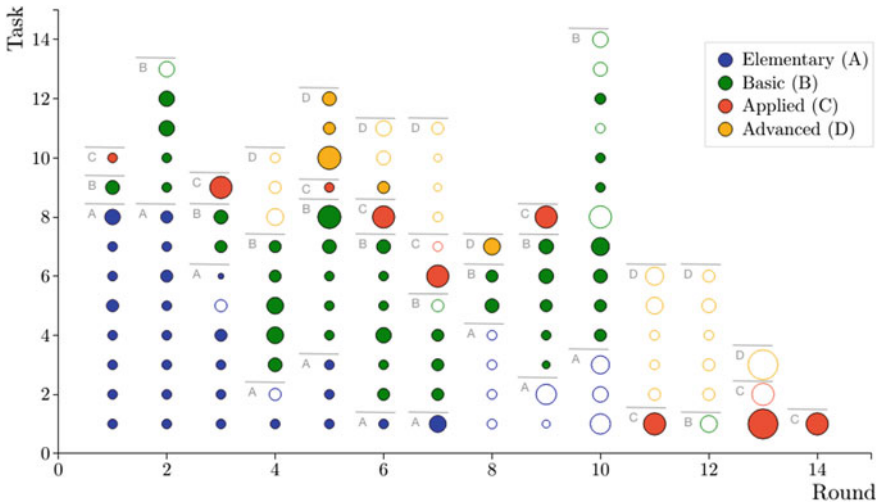
We have observed that there are students belonging in this group for two reasons: some of them co-operated all the time with a friend or a group of friends and some of them were regularly working in the multipurpose classroom waiting for the teaching assistant to solve all the problems they ran into. Nevertheless, the problem was the same for all of them: they did not learn the necessary programming skills to be able to work on their own and thus complete the programming assignment in the exam.

**Group 6: Dropouts by failing personal expectations:** These students were aiming at a high grade but collected enough points for passing at a lower grade only. They never took the exam of the course, and thus failed the course. It seems that they are ambitious and did not want to have a bad grade in their study register. These students most likely signed up for the course's next implementation.

**Group 7: Goal-driven students:** In the beginning of the course, these students completed as many assignments as possible. When they gained enough points for one of the categories, they immediately stopped working on assignments in that category. The students aiming at grade 1 could gain enough points for passing the course at around week 9. Figure 7 illustrates a course completion path for a student in this group who stopped working on the course assignments after week 9. Figure 8 illustrates a rather more complicated course completion path from a student representing this group. He stopped working on the assignments in different categories step by step.

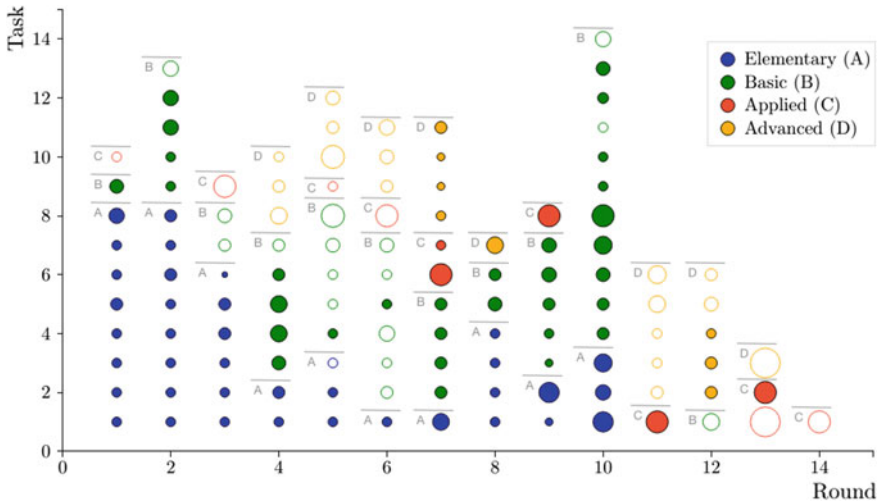


**Fig. 7** Goal-driven. Student who targeted grade 1 and achieved it. In the beginning of the course, he completed almost all the assignments. Then, on week 9 he gained enough of points for passing the course and stopped working on the course



**Fig. 8** Goal-driven. Student who targeted grade 3 and achieved it. He reached the requirements for A-points in week 7. After that he did not complete any A-assignments. Then, he reached the requirement for B-points in the middle of week 10 and did not complete any more B-assignments after that

The teaching assistants reported that some students were complaining that the assignments in the higher levels were too difficult to be completed without completing A-assignments related to the same topic but there was no need to complete the



**Fig. 9** Compensating. Student who targeted grade 3. However, he missed so many assignments that he only got grade 1. The biggest losses were the most difficult assignments in weeks 3–4 and almost all the assignments during weeks 5–6. Looking at the visualization quickly, it might seem that he has completed a lot of assignments. However, there are many large red circles (weeks 3, 5, and 6) that are not colored. He was grouped in the compensating students because he was able to catch up by completing some of the last assignments that were targeted for students targeting grades 3–5

A-assignments after fulfilling the A-point requirement. The complaining students were from this group and targeting grades 3–5. The complaints really encapsulate the aim of completing the programming assignments only to gain enough points. They did not see that completing the assignment in high-level categories would be easier if you had first completed the easy assignments related to the same topic.

**Group 8: Compensating students:** These students had a phase of the course where they missed a lot of deadlines or handed in a lot of solutions by the extended deadline and thus missed points. For many students, such an incident lead to dropping out from the course. However, the students in this group did not give up but patched up the missing points by extra assignments at the end of the course. Figure 9 illustrates one-course completion path of a student who was missing quite a lot of assignments in the first half of the course but did not dropout.

All the topics handled in the course build on the topics from the earlier weeks. This means that a student cannot just start working on the new assignment after missing all the assignments from the previous week(s). You always need some knowledge from the earlier weeks to complete the new assignments. Therefore, patching up the missed points can become very laborious.

The aim of the grading was that none of the assignments is so important that missing it would lead to failing the course. This grading system allows the students to compensate for missed assignments very liberally. However, compensating is not

easy so the students really pay for the free time they have taken in the middle of the course.

A student with less thorough knowledge is most likely not able to compensate for missing many assignments. We also observed multiple ways of dropping out from the course.

### **4.3 Student Feedback**

This student-centered, agile style of arranging the course that allowed the students to work according to their own schedule was praised by the students who completed the course. Approximately, 80% of students in each course implementation reported they worked “mostly independently” and over 10% “more often independently than in the multipurpose classroom” when asked for feedback after the course was over. The open-ended feedback from students mentioned often, e.g., “It is very effective to be able to work on one’s own pace”. Of course, there was also negative feedback from students who did not like the approach of working independently but most of the feedback was positive.

The students who were not majoring in computer science also gave positive feedback regarding the possibility to pass the course with less work. However, some students criticized the grading because they felt that it was unfair that you cannot have a good grade without completing the advanced assignments.

In the feedback, over half of the students checked that the course was too laborious. This, of course, was not our intention—rather, our idea was that the students work every week until their time for this course is up and then leave the rest of the assignments uncompleted. However, many of the students took more time for this course and completed most of the assignments even if they did not manage to do it in the time they were planning to. The teaching assistants reported that many students felt that they need to have grade 5 because they can decide the grade themselves. In the traditional way of grading, you typically do your best and hand in your solution for grading. Then the grader decides about your grade. Our way of grading turned this scene in the opposite direction: the students were able to decide their grade themselves by working for a longer time. Psychologically, there is a big difference in getting the grade that the grader decides for you and in making the decision yourself. For a student targeting grade 5, the decision to complete almost all the assignments had to be taken weekly again and again.

## **5 Discussion**

Fundamentally, our goal was to be able to meet the needs of different student groups. The results section gave us insights into student feedback, the grades they achieved, and different kinds of strategies for taking the course. While the feedback was gener-

ally positive and the students mostly liked the agile course setup, we also identified subjects for discussion and improvement.

### ***5.1 Criterion-Referenced Grading with Automated Feedback***

In the “traditional” way of grading, it is possible to define the requirements for each grade after all submissions have been received. In the criterion-referenced approach, the requirements have to be defined beforehand because they are the tool for guiding what students do. When this is combined with automated feedback, students know all the time what grades they have already earned and what grades are still reachable—provided they are able to pass the exam. After all, feedback students get from the assignments is one of the powerful influences on learning and achievement—either good or bad (Hattie & Timperley, 2007). In this study, we observed both sides of the feedback.

Automated assessment is sometimes criticized as it allows students to gather points from partially correct implementations. This can lead to a mismatch between actual and expected skills. In general, this could be tackled, for example, with a more coarse-grained grading of the assignments or by having some mandatory assignments where the feedback—even if automated—would be delayed (Spacco et al., 2006). Other approaches to addressing trial and error learning in the context of automatically assessed programming exercises are surveyed by Ihantola et al. (Ihantola, Ahoniemi, Karavirta, & Seppälä, 2010). We approached this challenge by grading nearly all assignments as passed/failed.

### ***5.2 (Un)Selecting the Assignments***

When looking at the strategies defined in the previous section, both *Opportunists* and *Goal-driven students* followed a clear strategy for optimizing their workload. This anticipated pattern is demonstrated by advantage taking students skipping the easy assignments. Goal-driven students, however, made a sudden stop before the end of the course when they had enough points for their target grade. They might have been able to pursue a higher grade but as the criterion-based grading scheme was publicly available, they decided to use the remaining time for something else—hopefully for other courses they felt more relevant for their further studies.

An alternative interpretation for the *goal-driven group* could be that after a certain point, the assignments simply became too difficult. The fact that some goal-driven students applied also the opportunist strategy at the same time, and that some of the stops were between grades supports this. Thus, we can ask, is it reasonable to allow students to choose which assignments are valuable for their learning? If the grading rules direct the student in the wrong direction, correcting the course of learning takes a lot of self-discipline and good self-regulatory skills.

Moreover, looking at the group's *Dropouts by missing assignments* and *Compensating students*, we perceive that there is a very fine line between passing and failing the course. This raises the question, is it fair for students to allow them to skip assignments if the consequences are troublesome: compensating work or failing the course entirely. We assume that the difference between *Compensating students* and *Dropouts by missing assignments* is closely linked to the self-regulatory skills of students. Thus, in the future, we should support such skills as well. For example, gamification and various visualizations affect self-regulation (Auvinen, 2015). In addition, peer review can boost self-efficacy (Zingaro, 2014), which contributes to the students' persistence to pass the course.

### 5.3 *Learning the Necessary Skills*

The group *Dropouts by missing skills* did well during the course but failed the exam. We assume that free riding in pair programming could have led to disparity between the exam and the assignments. Thus, we argue that Dropouts by missing skills could be helped by adding a midterm exam into the course requirements. This way they would recognize earlier that their way of working has led to problems with learning. If they recognized this early enough, it would still be possible to correct the problem before the final exam. Moreover, our electronic exam setting (Laine, Sipilä, Anderson, & Sydänheimo, 2016) allows students to take the exam in their own schedule, so it would be easy to add the midterm exam. We will consider it in the following course implementations. Another problem in our exam setup was that the students self-selected the difficulty of the exam. A longer exam where students would solve both easier and more demanding tasks should solve this problem.

### 5.4 *The Requirements of Passing the Course*

Finally, the group Dropouts by failing personal expectations is problematic from the teacher's perspective. The obligatory exam leaves students the possibility to decide to fail the course on purpose.

In our context, students can re-take this course and raise their grades later. The teachers, of course, wish that students would first make sure they pass and then later promote their grade if they still feel it is important. Thus, it seems that there is a need for redesigning the passing requirements so that it is not possible for the student to decide that he/she wants to fail the course like this. However, making the exam optional for grade 1 is not ideal, since the existence of the group Dropouts by missing skills proves that there is a need for a controlled exam. Midterm exam, discussed in the previous section, and improved (automated) detection of extensive collaboration (Hellas, Leinonen, & Ihanntola, 2017; Yan, McKeown, Sahami, & Piech, 2018) are the alternatives we consider to address these challenges in the future.

## 5.5 *Over Performing*

In our study, the group Perfectionists did extra on the A-level assignments as well as over-performed by completing assignments where they learned new and exciting things (level C and D in our case).

There can, of course, be explanations as to why students want to complete the voluntary A-assignments: Some of them do not have prior programming experience and thus need to learn all the details. The A-assignments are designed for this and thus useful. It is also possible that some students completed them at the beginning of the course just to do something since there were not many C- and D-level assignments at this phase of the course. However, in the later part of the course, there were also A-assignments for the students who had difficulties in the course, and thus missed some assignments here and there. These A-assignments are repeating the basic concepts that have already been learned in the earlier assignments. Therefore, we assume that a student who is going to have grade 5 will not learn much by completing these repetitive extra A-assignments. To our understanding, the only motivation of this student group for completing these repetitive extra A-assignments is that they want to have the maximum points on the course platform. Thus, we decided to name them the Perfectionists.

## 5.6 *Summarizing Different Student Strategies*

The course completion strategies resemble the strategies identified by Karavirta, Korhonen, and Malmi (2005). Based on the clustering of how students use resubmits in the context of automated assessment, they divide learners between passers, ordinarys, iterators, ambitious, and talented. One can argue that our Perfectionists are also ambitious. Advantage taking students are most likely talented, and both compensating and goal-driven students could belong into passers. In the future, it would be extremely interesting to combine the information from resubmissions to learning paths and use the improved profiling for adaptive learning (Brusilovsky et al., 1998).

## 5.7 *Further Observations*

Finally, it can be regarded as unfair that the students, who fail to complete advanced assignments, cannot get grade 5. However, we believe it is also important to learn to set priorities, which is an important work-life skill for the later career—if you are not majoring in CS do you need to have the best grade in a laborious programming

course? Granted, for a more homogeneous student group, such considerations might be misleading, but based on experience, students, in any case, set different priorities to their courses, and hence offering an agile, well-defined approach to do so has been regarded welcome.

## 6 Conclusions

In this chapter, we have provided an insight into the way that we provide personalized learning experiences in a university-wide introductory programming course. Course content is flexibly defined by the learners, and, based on the individual goal setting, the students are allowed to choose the assignments they work on. In analogy to agile software development, this corresponds to allowing the development team to decide which features they choose to work on, depending on the complexity and other factors associated with the feature. We have also shed light on how students experience and utilize the agile course setup by analyzing their study paths. In this respect, the following research questions directed our study.

1. How are students' self-reported target grades in the beginning of the course related to the final learning outcomes?

Just setting a goal—in our case, the target grade—seems to have a strong positive correlation with passing the course. Moreover, there is a small/medium correlation between the target grade and final grade. Interestingly, the means of the target grades in Autumn 2016 (including CS students) and Spring 2017 (mainly other students) were nearly the same, while the actual performance in Autumn 2016 was much better (see histograms in Fig. 4). This hints that the presumably weaker students did not utilize the grading scheme as we expected. Perhaps it is difficult to set yourself low expectations. Given the way that the course was designed, this may have guided weaker students to do tasks that were too difficult.

2. What kinds of behavioral patterns can be detected among students in the agile course setup?

The main objective of this study was to understand how students utilize the agile course and how it works from the perspective of the diverse student population. However, in the process, we identified three distinct behavioral patterns that were related to high performance (i.e., Perfectionists, Opportunists, and Casual well-performers), three patterns of failing the course (i.e., Dropouts by missing assignments, by missing skills, and by failing personal expectations), and two other behavioral patterns associated with how students work (i.e., Goal-driven students and Compensating students).

The analysis of the goals and the perceived behavioral patterns show that there are various ways of completing the course. Unfortunately, we also perceived various ways of failing the course which suggests that the low performing students should be taken more into consideration when developing the course. Notwithstanding,



the diversity of the students' strategies demonstrates that the criterion-referenced grading scheme certainly supports the diversity of the student population better than a traditional course setup where all students complete the course in the same manner. Adjustment of the grading criterion is needed though.

Obviously, there are numerous directions for future work. In fact, almost any of the identified student groups could be more elaborately studied to understand their motives and goals. In particular, understanding why some talented and clearly capable students also complete simpler assignments rather than directly focusing on more complex problems is an interesting issue we plan to address in the future. In addition, studying how the students advance in their studies later on, and whether this correlates with their results and motivations in this course forms an interesting piece of further research.

## References

- Ahadi, A., Lister, R., Haapala, H., & Vihavainen, A. (2015, July). Exploring machine learning methods to automatically identify students in need of assistance. In *Proceedings of the Eleventh Annual International Conference on International Computing Education Research* (pp. 121–130). ACM.
- Auvinen, T. (2015). Educational technologies for supporting self-regulated learning in online learning environments. Doctoral dissertation. Retrieved from <http://urn.fi/URN>. ISBN:978-952-60-6281-5.
- Biggs, J., & Tang, C. (2007). *Teaching for quality learning at university* (3rd ed.). Open University Press.
- Bishop, J. L., & Verleger, M. A. (2013, June). The flipped classroom: A survey of the research. In *ASEE National Conference Proceedings, Atlanta, GA, USA* (Vol. 30, No. 9, pp. 1–18).
- Bloom, B. S., Engelhart, M. D., Furst, E. J., Hill, W. H., Krathwohl, D. R., et al. (1956). *Taxonomy of educational objectives: The classification of educational goals. Handbook I: Cognitive domain*. New York: David McKay Company, Inc. (7th Edition 1972).
- Brusilovsky, P., et al. (1998). Adaptive educational systems on the world-wide-web: A review of available technologies. In *Proceedings of Workshop "www-Based Tutoring" at 4th International Conference on Intelligent Tutoring Systems (ITS'98), San Antonio, TX, USA*.
- Carter, J., Ala-Mutka, K., Fuller, U., Dick, M., English, J., Fone, W., et al. (2003). How shall we assess this? In *ACM SIGCSE Bulletin* (Vol. 35, pp. 107–123).
- Carter, J., Bouvier, D., Cardell-Oliver, R., Hamilton, M., Kurkovsky, S., Markham, S., et al. (2011). Motivating all our students? In *Proceedings of the 16th Annual Conference Reports on Innovation and Technology in Computer Science Education-Working Group Reports (ITiCSE'11)* (pp. 1–18). ACM.
- Carter, J., White, S., Fraser, K., Kurkovsky, S., McCreesh, C., & Wieck, M. (2010). ITiCSE 2010 working group report motivating our top students. In *Proceedings of the 2010 ITiCSE Working Group Reports* (pp. 29–47). ACM.
- Douce, C., Livingstone, D., & Orwell, J. (2005). Automatic test-based assessment of programming: A review. *Journal on Educational Resources in Computing (JERIC)*, 5(3), 4.
- Fuller, U., Johnson, C. G., Ahoniemi, T., Cukierman, D., Hernan-Losada, I., Jackova, J., ..., Thompson, E. (2007, December). Developing a computer science-specific learning taxonomy. *SIGCSE Bulletin*, 39(4), 152–170.
- Hattie, J., & Timperley, H. (2007). The power of feedback. *Review of Educational Research*, 77(1), 81–112.

- Hellas, A., Leinonen, J., & Ihanola, P. (2017). Plagiarism in take-home exams: Help-seeking, collaboration, and systematic cheating. In *Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education* (pp. 238–243). ACM.
- Ihanola, P., Ahoniemi, T., Karavirta, V., & Seppälä, O. (2010). Review of recent systems for automatic assessment of programming assignments. In *Proceedings of the 10th Koli Calling International Conference on Computing Education Research* (pp. 86–93).
- Illeris, K. (2002). *The three dimensions of learning*. Malabar, Florida: Krieger Publishing Company.
- Johnson, C. G., & Fuller, U. (2006). Is Bloom's taxonomy appropriate for computer science? In *Proceedings of the 6th Baltic Sea Conference on Computing Education Research: Koli Calling 2006* (pp. 120–123).
- Johnson, G., Gaspar, A., Boyer, N., Bennett, C., & Armitage, W. (2012). Applying the revised Bloom's taxonomy of the cognitive domain to Linux system administration assessments. *Journal of Computing Sciences in Colleges*, 28(2), 238–247.
- Karavirta, V., Ihanola, P., & Koskinen, T. (2013, July). Service-oriented approach to improve interoperability of e-learning systems. In *2013 IEEE 13th International Conference on Advanced Learning Technologies (ICALT)* (pp. 341–345). IEEE.
- Karavirta, V., Korhonen, A., & Malmi, L. (2005). Different learners need different resubmission policies in automatic assessment systems. In *Proceedings of the 5th Annual Finnish/Baltic Sea Conference on Computer Science Education* (pp. 95–102).
- Keim, D. A. (2002). Information visualization and visual data mining. *IEEE Transactions on Visualization and Computer Graphics*, 8(1), 1–8.
- Laine, K., Sipilä, E., Anderson, M., & Sydänheimo, L. (2016, 9). Electronic exam in electronics studies. In *SEFI Annual Conference 2016*.
- Lister, R., & Leaney, J. (2003). Introductory programming, criterion-referencing, and bloom. *ACM SIGCSE Bulletin*, 35(1), 143–147.
- Mazza, R., & Milani, C. (2005). Exploring usage analysis in learning systems: Gaining insights from visualisations. In *Workshop on Usage Analysis in Learning Systems at 12th International Conference on Artificial Intelligence in Education* (pp. 65–72).
- Merriam, S. (2001). Andragogy and self-directed learning: Pillars of adult learning theory. *New Directions for Adult and Continuing Education*, 89, 3–14.
- Romero, C., & Ventura, S. (2010). Educational data mining: A review of the state of the art. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 40(6), 601–618.
- Seddon, G. M. (1978). The properties of Bloom's taxonomy of educational objectives for the cognitive domain. *Review of Educational Research*, 48(2), 303–323.
- Spacco, J., Hovemeyer, D., Pugh, W., Emad, F., Hollingsworth, J. K., & Padua-Perez, N. (2006). Experiences with Marmoset: Designing and using an advanced submission and testing system for programming courses. *ACM SIGCSE Bulletin*, 38(3), 13–17.
- Thompson, E., Luxton-Reilly, A., Whalley, J. L., Hu, M., & Robbins, P. (2008). Bloom's taxonomy for CS assessment. In *Proceedings of the Tenth Conference on Australasian Computing Education Conference* (Vol. 78, pp. 155–161).
- Yan, L., McKeown, N., Sahami, M., & Piech, C. (2018). TMOSS: Using intermediate assignment work to understand excessive collaboration in large classes. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education* (pp. 110–115).
- Zingaro, D. (2014). Peer instruction contributes to self-efficacy in CS1. In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education* (pp. 373–378).

# Teaching and Fostering Reflection in Software Engineering Project Courses



Håkan Burden and Jan-Philipp Steghöfer

**Abstract** Reflection is an important part of agile software processes as witnessed, e.g., by the Sprint Retrospectives in Scrum or by the various learning feedback loops in XP. Engineering education also emphasizes the importance of reflective practice, e.g., in Kolb's learning cycle and Schön's reflection-in/on-action. Our contribution in this chapter is a toolkit for reflective practice that shows how reflection can be used by software engineering students for two purposes: to reflect on the application of a software process and to reflect on their learning process. In order to help students understand the purpose of reflection and how to approach reflection, we follow a cognitive apprenticeship approach in which the teachers reflect about the events in the course, their own goals, and how they are aligned with the teaching. Students are asked to reflect during supervisions and as part of their written assignments from the very beginning of the course. We thus combine a meta-cognitive approach where reflection is taught as a learning strategy with a common software engineering practice of continuous improvement through reflection. We evaluate the reflective model and a course design based on it through the student, teacher, and theoretical lenses based on empirical data.

**Keywords** Agile · Scrum · Computer science education · Software engineering Project course · Reflective practice

---

H. Burden (✉)  
RISE Viktoria, Gothenburg, Sweden  
e-mail: [hakan.burden@ri.se](mailto:hakan.burden@ri.se)

H. Burden · J.-P. Steghöfer (✉)  
Chalmers | University of Gothenburg, Gothenburg, Sweden  
e-mail: [jan-philipp.steghofer@cse.gu.se](mailto:jan-philipp.steghofer@cse.gu.se)

© Springer Nature Singapore Pte Ltd. 2019  
D. Parsons and K. MacCallum (eds.), *Agile and Lean Concepts for Teaching and Learning*, [https://doi.org/10.1007/978-981-13-2751-3\\_12](https://doi.org/10.1007/978-981-13-2751-3_12)

# 1 Introduction

Reflective practice is to evaluate your own actions and their consequences to engage in a process of continuous learning and is therefore an essential ability for professional development (Brookfield, 1995; Lyons, 2010). It enables us to not only learn from our experiences but to grow as professionals since reflection helps us challenge our assumptions and develop new professional skills as well as meta-cognitive strategies which will help us make informed decisions even when time and resources are scarce (Schön, 1983).

Despite the known benefits of reflection for professional development, there is a lack of attention within engineering education on integrating reflective practice in both courses and educational programs (Turns, Sattler, Yasuhara, Borgford-Parnell, & Atman, 2014). One of the reasons is that reflection can be intimidating since it is often perceived as sharing private thoughts and even shortcomings (Gunn, 2010). It is also challenging in the sense that reflection in some way or other asks the question of what could have been done differently. The third obstacle for teaching and learning reflective practice is that there is no clear definition of what reflection actually is.

We, therefore, explore reflective practice—from both a student and a teacher perspective—within an engineering project course and provide answers to two research questions:

**RQ 1** How can we facilitate reflective practice in a software engineering project course?

**RQ 2** How do students utilize opportunities for reflective practice for their continuous learning?

We will answer these research questions in the context of the development of a software engineering project course, which also shows how the outcomes directly inform a course design.

*Our contribution* is a toolkit for reflective practice, an artifact based on our strategy to plan, perform, and assess reflection in our course.

*This chapter* is structured so that first, Sect. 2 details the theoretical framework and related work and Sect. 3 describes action design research, the methodology we have chosen for our own course development. The situation as it stood when we started teaching the course is described in Sect. 4 which is then followed in Sect. 5 by a description of the toolkit for reflective action we derived from applying our methodology to the challenges we faced. Section 6 details the changes to the course after applying the toolkit. We then share both our own and the student reflections in Sect. 7 before we broaden the scope in Sect. 8 to look at how the toolkit could be applied outside software engineering. Finally, the last section concludes our reflection and points to future work.

## 2 Background

In this section, we will discuss the role of reflection in software engineering education and in software engineering practice as a starting point for our proposal to introduce reflection as a mainstay in software engineering project courses.

### 2.1 Reflection and Education

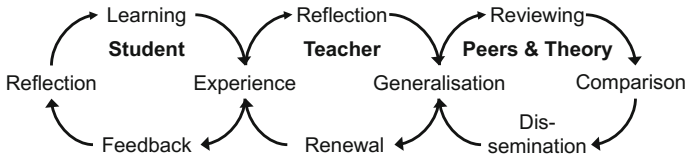
There are numerous definitions of reflection in the educational literature. Shkedi states that “*Reflection is meta-thinking (thinking about thinking) in which we consider the relationship between our thoughts and our actions in a particular context*” (2000). Smith defines reflection as “*What is in relation to what might or should be and includes feedback to reduce the gap*” (2001) while Mann defines that “*reflection is a process of inner dialogue*” (2005). Loughran summarizes the situation by stating that “*for some [reflection] simply means thinking about something, whereas for others it is a well-defined and crafted practice that carries very specific meaning and associated action*” (2002).

Schön—in describing the *reflective practitioner*—distinguishes between reflection-in-action and reflection-on-action (1983). Reflection-in-action is what we do when we encounter new situations and need to improvise on how to best proceed. Reflection-on-action is when we, later on, have an opportunity to sit down and go through the experience again in our minds to assess how it went and see what we could have done differently. On the same bearing, Freire defines *praxis* as the balance between theory and action where reflection is a means to achieve practices grounded in reflection (2000). He further states that praxis is not easy as it requires “*wise and prudent practical judgement about how to act in this situation*”.

Kolb’s learning cycle (2014) ties concrete experiences in a specific context to conceptualized insights from reflective practice. The transformation is facilitated by reflecting on one’s own experiences to modify known concepts, generate new hypotheses and seek out what others have to say about similar situations. The gained insights are then used to set up new experiments where the hypotheses can be tested and new experiences gained.

However, applying Kolb’s learning cycle will yield different results depending on who does it. Brookfield refers to this by using different lenses, where applying a new lens gives new insights (1995). For an educational setting, Brookfield defines four lenses to structure reflection for the teacher. The first lens is the autobiographical lens which is used to reflect from a personal point of view. The second lens is the student’s perspective while the third lens is that of the teacher’s peers. Finally, the fourth lens represents relevant theory and proven practice.

Cognitive apprenticeship (Brown, Collins, & Duguid, 1989) can be seen as an attempt to reason around the two lenses of student and teacher from the well-known concept of apprenticeship. But instead of a setting where the craft has a central role,



**Fig. 1** The triple learning cycle, adapted from Elmgren and Henriksson (2010). Student and teacher learning is interconnected

the focus is on the cognitive skills needed for higher education. The roles of novice and master are now substituted for the roles of teacher and student as the teacher uses different techniques for the students to fathom the knowledge and skills of the master. There are six techniques:

**Modeling:** The master demonstrates explicitly how a task is done.

**Coaching:** The master supervises the novice in carrying out a task.

**Scaffolding:** The master sets up supportive structures to guide and help the student to experiment on their own.

**Articulation:** The novice uses the terminology of the trade to express new insights, their reasoning and formulate new challenges.

**Reflection:** The novice is given the opportunity to compare their own knowledge and skills in relation to those of the master.

**Exploration:** The novice is given the freedom to explore on his or her own without the interference of the master.

Being the master in such a context requires to both assess in advance how to conduct the different techniques but also to reflect-in-action to adjust them, such as in the case of scaffolding or coaching.

Inspired by Kolb, Elmgren and Henriksson (2010) represent Brookfield's four lenses in a triple learning cycle (cf. Fig. 1) where the students and the teachers meet in the concrete experience and the teacher's generalizations are shared and reviewed by peers. In this way, each cycle is informed by other cycles and knowledge is shared and spread beyond the personal cycle.

Apart from the aspects already mentioned, the literature referenced above reveals two more interesting takeaways: (1) reflection requires doing, be it in terms of experimentation (Kolb, 2014), action (Freire, 2000) or exploration (Brown et al., 1989); and (2) reflection is contextual and varies over time (Schön, 1983) and person (Brookfield, 1995).

## 2.2 Reflection in Software Engineering

Reflective practice is an essential aspect of software process improvement (SPI). The classic SPI loop championed, e.g., by Villalón et al. (2002), consists of the four

steps “evaluation of the current situation,” “plan for improvement”, “implement the improvements,” and “evaluate the effect of the improvements” and thus constitutes a classic reflection loop as discussed above. Most SPI methods are built around this notion and contain approaches for some or all of these steps. For instance, inductive approaches like quality improvement paradigm (QPI)/Experience Factory (1993, 1995) and iFLAP (2008) focus on the evaluation of the current situation, the derivation of goals, and the creation of measurement plans to determine whether any changes were successful. Descriptive approaches like CMMI (2010) on the other hand focus on the planning of concrete improvement steps.

Since a structured improvement effort using one of the methods mentioned above is usually associated with a dedicated effort on the organization level and significant resources are contributed to it, long-term strategic goals are the focus of SPI (Huber, 1996). Reflection, therefore, takes place on an abstract level that encompasses a larger part of the organization and several software development efforts. The goals are often to increase long-term productivity, quality, or responsiveness to customer needs.

In contrast, iterative processes, in particular, agile ones, have a strong focus on constant change and improvement (Williams & Cockburn, 2003) and allow embedding a similar loop into each iteration and even in daily activities. An overview of such activities is provided by Babb (2014). They include group programming where reflection-in-action is practised by the people working together, estimation and planning activities, as well as the daily standups that many agile teams perform. In these cases, the reflection is implicit in the interaction of the team members and thus constitutes a social effort in communities of practice (Wenger, 1998).

Reflection is most explicitly practised in *sprint or iteration retrospectives* that are used to allow the team to discuss the current situation, identify desired future states, and devise a plan to get there. This shortens the round-trip time for improvements considerably and allows the developers to quickly try out and evaluate improvement ideas that can be helpful in the short term. Three questions are usually addressed in these short meetings: What worked well for us? What did not work well for us? What actions can we take to improve our process going forward? These questions focus the team on rather immediate issues. The reflection is therefore mostly tactical. This is in the agile spirit where immediate benefits, responding to changes, and short-term wins are emphasized over long-term strategy. However, even in such an environment, retrospectives with a broader scope that stretch all iterations for one release have been suggested as a way to reflect on the longer-term planning (Maham, 2008).

In situations in which iterations are not used (e.g., in classic waterfall projects or in projects using the V-model prominent in the automotive industry), *post-mortem reviews* are a tool that allows the development team to reflect on their work (Dingsøyr, 2005). In contrast to iteration retrospectives, postmortems have a broader scope that goes beyond a single iteration, are usually more formal and more involved, and emphasize organizational learning (Dingsøyr, 2005). Combining several post-mortems from the same organization can even help to reflect on high-level management practices that influence the effectiveness of all software development efforts in the company (Dingsøyr, Moe, Schalken, & Stålhane, 2007).

Even though the reflective practice is widespread and its positive impact is reported and empirically validated, the concrete design of the reflective activities need to be carefully tuned to achieve the desired results. The accuracy of effort estimations, e.g., is a common problem in software development projects. It was shown that it does not improve if the engineers that estimated the efforts reflected on their estimations themselves, but that improvements are only made if other professionals provide feedback (Jørgensen & Gruschke, 2009). This indicates that reflection needs to be explicitly fostered and “engineered” in order to be effective in practice.

### 3 Methodology

We use a modified version of *action design research* (Sein, Henfridsson, Purao, Rossi, & Lindgren, 2011), a combination of action research and design science research that focuses on designing and evaluating artifacts in a situation that “is dependent on the interaction of the participants of the research” and “can only be performed in the context of the organization and with the involvement of people within the environment under study.” (Dresch, Lacerda, & Antunes, 2014, p. 94). Our modifications target the type of artifact created: instead of an ensemble of IT artifacts as described by Sein et al. (2011), we create an ensemble of teaching artifacts. This ensemble constitutes our toolkit for reflective practice in SE education (cf. Sect. 5). We describe the main characteristics and steps of action design research and how we implemented them in the following.

#### 3.1 Action Design Research Applied to Education

Action design research is characterized by how it engages with the organization the subject of the research is embedded in. In particular, it is suitable for situations in a specific organizational setting that are addressed by intervention and evaluation within this setting. This maps very well to the educational context since we address situations that are dependent on the specific settings of the course, the students, the program, the university, etc. Solutions to challenges observed in specific courses thus constitute the artifacts that are the outcome of the methodology. They address this situation and have to be evaluated within it. The artifact is thus not only an outcome achieved by the knowledge and expertise of the researchers but heavily influenced by the users (the students and teachers in this case) and the dynamically changing situation it is designed for and evaluated in.

We follow the four stages and the associated principles suggested by Sein et al. (2011), mapping each to the educational context to which we apply the methodology. In the following, we will describe these stages and how we mapped them to our situation.



*Stage 1: Problem Formulation* The intended outcome of this stage is the framed problem and the theoretical premise. The two principles to follow prescribed by Sein et al. (2011) are that the research should be *practice-inspired* and that the artifact should be *theory-ingrained*. We found ourselves in a situation where there was a gap between the course's intended learning outcomes and the tools available to achieve them. This was clearly a practically relevant problem since it affected our work as teachers as well as the learning process of the students. Constructive alignment (Biggs, 1996) served as a theoretical foundation to evaluate this gap and the principle of the reflective practitioner (Schön, 1983) informed our first ideas for a solution approach. An additional, important practical component was that both involved teachers had a long-term commitment to the course and were thus not only interested in improving it in several iterations but could also expend the necessary resources to do so.

*Stage 2: Building, Intervention, and Evaluation* This stage's intended outcome is a realized (educational) artifact that has been evaluated and refined by use in the relevant situation. It is based on the problem formulation from the first stage. In this stage, we developed the artifact, a toolkit of reflective practices for the software engineering project to improve the course's constructive alignment. The artifact was deployed and evaluated in several iterations of the course and refined after each iteration. We followed the principle of *reciprocal shaping* since the educational artifact shapes the work with the students in the classroom which in turn has an influence on the artifact. We also ensured the *mutual learning* principle by giving the students the chance to learn from us and our attempts to use the toolkit and using the feedback from the students to learn about the effectiveness of the different tools. Finally, we adhered to the *authentic and concurrent evaluation* principle by evaluating the interventions immediately in an authentic setting.

*Stage 3: Reflection and Learning* The intended outcome of the reflection and learning stage is a generalized artifact that applies "to a broader class of problems" (Sein et al., 2011) achieved within a continuous learning cycle. In our context, that meant abstracting the developed toolkit from the specifics of the course and making it applicable to different course settings. We achieved this through continuous evaluation of the results in the classroom w.r.t. our goals and constructive alignment. Generality was achieved by identifying the specific issues in the course, separating them from the abstract concepts and ideas we applied, and using these concepts to identify new approaches (i.e., new tools to add to our toolkit). The artifact thus emerged guided by our reflection of the evaluation results, fulfilling the principle of *guided emergence*.

*Stage 4: Formalization of Learning* This stage builds upon the previous one by abstracting the generalized artifact and the problem further into design principles and the characteristics of a problem class. This stage thus allowed us to move the toolkit from the specific course instance into a broader context and abstract it to become usable for other teachers. We generalized the outcomes of the specific course to formalize the toolkit as described in Sect. 5. The abstract toolkit was then instantiated for the course iteration in spring 2017 and evaluated there to show its viability. Our final

result includes a generalization of the problem instance (software engineering project courses), a generalization of solution (toolkit), and design principles (model aspect of toolkit), thus following the *generalized outcomes* principle. The formalization of the results for dissemination is represented by this publication.

### 3.2 Data Collection and Analysis

In order to formulate the problem (stage 1) and to reflect and learn (stage 2) from our experience, we used a number of data sources that we analyzed repeatedly, mostly to derive qualitative data about the effectiveness of the toolkit for reflective practice and thus our teaching approach. While we evaluated data from all course iterations between autumn 2014 and spring 2017, our discussion in this chapter is focused on the latter iteration and the insights we gain from the evaluation of our current, stable version of the developed artifact.

*Course evaluations* We used the course evaluations conducted by the university as a tool to gauge the satisfaction of the students and identify issues with constructive alignment, workload, and cognitive demand. The evaluations consist of a voluntary, anonymous, web-based survey among the students, and a meeting with student representatives from the course itself as well as from the student union that is led by the course coordinator. We thus had the results of the survey as well as the notes from the evaluation meetings as a basis for our own analysis. We focused the analysis of the evaluation results on course development (Edström, 2008) and used it to understand which aspects the students struggled with and needed to be addressed better. As such, we attempted to receive formative feedback from the students, in particular through the discussions at the meeting.

*Student reports* A further source of information were the reports the students wrote throughout the course. There are three mandatory written hand-ins: the students need to reflect on how they defined their process, they need to give a report on their progress by half-time of the course, including how they refined the process, and they need to describe their overall process and lessons learned at the end of the course. In many cases, the students refer to lessons they transferred from specific teaching moments. In addition, the feedback the students got from the teachers on their reports contains connections between the reports and the individual teaching moments. We use this information to check if our toolkit is constructively aligned and yields the desired outcomes. We analyzed the reports quantitatively, applying a lightweight coding in a separate session that was independent of the grading.

*Teacher notes* Whenever we introduced a new teaching moment (i.e., a new tool in our reflective toolkit), we took extensive notes about the reaction of the students, whether or not we feel we achieved our objective, which questions students asked, if the time allotted was sufficient, etc. Usually, there were two sets of notes available, but occasionally only one teacher could be present during the introduction of a new tool and we relied on his notes in this case.

### 3.3 Threats to Validity

We discuss threats to the validity of our study and the methods used to minimize the threats, following the classification in Tomal (2010). While this classification was intended for action research, it applies well to action design research since it has a strong focus on the participants of the study, i.e., the organizational setting in which the research takes place.

One of the main threats is *differential selection*, i.e., collecting and comparing data from different groups of students in the different iterations of the course. Indeed, in different course iterations, there are changing proportions of students from different programs. In the spring iterations, most of the students come from the program on Industrial Economy, while in the autumn, most of the students are from Information Technology or Computer Engineering. The different backgrounds cause differences in how the students perceive different teaching moments and which expectations they have coming into the course. While students with a computer science background, e.g., expect a stronger technical focus and are surprised by the focus on process and reflection, industrial economy students can feel overwhelmed by the programming tasks. We addressed this threat by leveraging the longitudinal aspect of our study and trying out our tools in both settings for increased generality.

A related threat is that of *history*, i.e., differences when data is collected at different points in time. This is certainly an issue here since we combine data from different course iterations. However, we have mitigated this threat since we evaluated the data directly after the course instance in order to develop the course and our toolkit further.

*Contamination*, i.e., unaccounted factors outside of the study influencing its result, can be a factor here. For instance, a persistent student complaint is that the scheduling of students of different programs is not compatible, making it hard for mixed groups to find time to work together. However, due to the considerable experience of the teachers, we have a good overview of the course environment and can take such factors into account.

The threat of *instrumentation*, i.e., influences of the data collection method, can play a role, in particular since graded material was used. However, all data sources were always cross-referenced and none used in isolation. In particular, the anonymous course evaluation survey reduced the threat of bias. However, since the different data sources capture different kinds of data, a residual effect might remain.

Finally, the threat of *researcher bias* has been addressed by planning, designing, acting, and evaluating as a team. While it is possible that the team as a whole has a bias, the two teachers provide complementary viewpoints and approaches. In addition, there has been continuous exchange with program managers, the student union, students in the course, and other teachers about the course and the different attempts made to improve it.

Furthermore, there are a number of potential threats to validity that were not observed in our study: *Attrition*, i.e., the loss of participants while the study was ongoing, was not a major issue in this study since almost all students that enrolled in the course instances finished them. The *Hawthorne effect*, i.e., participants perform

better since they are given attention, is also negligible since our data collection methods are nonintrusive and only use elements that occur in the normal progress of the course anyway. While there is *maturation* of the participants during each course instance as an effect of the teaching, this is not a major concern for our study since each course instance started with a new set of students with little to no carryover from previous instances. The threat of *testing*, i.e., participants learning from pretests and thus answering differently in posttests, was also not an issue since our data sources did not include such tests. There might be a learning effect from the different student reports based on the feedback from the teachers, but this effect is intentional.

## 4 The Old Course Design

The starting point for our endeavor is the course instance in which the authors were first involved in the autumn of 2014. The Software Engineering Project Course represents 7.5 ECTS or 10 weeks of half-time studies and was taken by 173 students from 3 different bachelor programs run by the Computer science and Engineering department, which is a shared department of Chalmers University of Technology and the University of Gothenburg. The students formed 29 teams and collaborated with an external stakeholder in developing Android apps for truck drivers which were safe to use while driving.

The rest of this section is organized following a constructive alignment perspective where we first introduce the intended learning outcomes (ILO), learning activities and assessment tasks (Biggs, 1996) before we discuss the benefits and shortcomings of the course from a teacher and a student perspective.

### 4.1 *Intended Learning Outcomes*

The course's intended learning outcomes reflect the ambition to give an overview of what canonical software engineering is as a subject area (Burden, 2017), for instance, as defined in the Software Engineering Body of Knowledge [SWEBOK, (Bourque et al. 2014)]. Thus, the course aims regarding knowledge and understanding state that the student should be able to...

- ILO1** ...identify the complexities of software design and development,
- ILO2** ...describe the fundamentals of software engineering, such as stakeholders and requirements, and
- ILO3** ...describe the difference between the Customer, the Solution, and the Endeavor as well as the different methods used for each

after successfully finishing the course. In terms of skills and abilities, the student should be able to...

- ILO4** ...elicitate requirements from and design a solution to a real-world problem,
- ILO5** ...plan and execute a small software development project in a team,
- ILO6** ...apply skills from programming and other relevant courses, as well as
- ILO7** ...learn new tools and APIs on his/her own.

Finally, the students are also expected to be able to...

- ILO8** ...reflect on the choice of software engineering methods used throughout the project.

Following Bloom's revised taxonomy (Anderson, Krathwohl, & Bloom, 2001), the first three ILOs revolve around factual and conceptual knowledge such as basic terminology and how these relate to each other. The rest of the ILOs focus on procedural knowledge such as methods and procedures within the SE domain. The exception is ILO7 which is meta-cognitive since it requires the students to reason around their own learning.

## 4.2 *Learning Activities*

The learning activities consisted of lectures and supervision with a final presentation at the end of the course. The supervision was run on a weekly basis and students who had already taken the course were paid to supervise. The content of the supervision was supposed to target the process aspects that the student teams encountered but often revolved around tool and technology issues, such as git merging or Android debugging. There were 13 lectures which included a lecture each to introduce the course and the project scope, 4 lectures on the project-specific tools and technologies, 2 lectures on software engineering in general and 2 lectures on Scrum, 1 guest lecture, and a lecture on which tests and documentation the teams were supposed to hand in. Since the students were supposed to reflect on their choice of methods and practices but never had been given the opportunity to get feedback on their reflections, a final lecture was added while the course was running to give the students an idea of what reflection could be. During the reflection lecture, one of the authors presented his own reflections on how the course had panned out and what could be done differently for the next course instance.

The project started the same week as the course so that the second lecture introduced the project scope. This meant that the lectures regarding Scrum were given after the students had started their development effort. Subsequently, they had to make large changes to how they worked or disregard Scrum to continue working in an ad-hoc manner.

### 4.3 Assessment

The assessment was purely summative in terms of teacher engagement and consisted of five major elements:

**Vision** 30% of the grade was determined by how well the product matched the vision, the stability of the product, and the user experience;

**Design** Design decisions and how these were documented accounted for 10% of the final grade;

**Code** The code quality and the technical complexity of the solution made up 15% of the final grade;

**Tests** A further 15% of the final grade depended on which tests had been done and the documentation of the product;

**PMR** Finally, 30% of the final grade was based on a postmortem report written after the final presentation.

The five elements assessed the team performance. To be able to give the student's individual grades, the students were also asked to fill out a personal evaluation for each team member. Together with a summary of who had contributed what to the code base, this enabled the teachers to give individual students a grade that differed from the team's overall grade.

### 4.4 Constructive Alignment and Student Perception

The relationship between the ILOs, learning activities, and assessment tasks are visualized in Table 1. Activities and tasks can overlap: the lecture on how to use Android, e.g., relates both to ILO6 and ILO7 since it both offer an opportunity to apply skills from previous courses and learn new technologies. In the same way, the project was used to identify the complexities of software development (ILO1) and give an opportunity to execute a small software project in a team (ILO5). A shortcoming of the old course design was how the student teams needed to find ways

**Table 1** The course alignment matrix for the old course design

ILO	Learning activity	Assessment tasks
ILO1	11 lectures and project	7 supervisions and terminology
ILO2	11 lectures and project	7 supervisions and terminology
ILO3	8 lectures and project	7 supervisions, vision, design and PMR
ILO4	6 lectures and project	7 supervisions, vision, design and PMR
ILO5	3 lectures and project	7 supervisions and PMR
ILO6	5 lectures and project	7 supervisions, design, code and tests
ILO7	5 lectures and project	7 supervisions, design, code and tests
ILO8	1 lecture and project	7 supervisions and PMR

to transfer the theoretical content of the lectures into practical skills during the project themselves. This was also remarked upon in the course evaluation and strategies to carry out the transfer were requested by the students. Regarding ILO7—students should be able to learn new tools and APIs on their own—the content of the five lectures centered around demonstrating how to configure the tools and make calls to the API. During the course, it became obvious that the students struggled with reflecting on their process and the decisions they took. Therefore, one of the teachers decided to add a final lecture to the schedule where he reflected himself upon the design of the course and what he would do differently if he had the opportunity to give the course again.

Regarding assessment, the only opportunity for formative feedback was during the weekly supervision slots. This relied on the student supervisors to be present and capable of handling process-related discussions. The course evaluations indicate that a recurring problem was that half of the supervisors were difficult or impossible to get in touch with and that those who carried out their supervision focused on tool related issues: *“The TA was not involved [ , did not have] enough knowledge of the course or helped us in any way”*. There were no assessment tasks directly related to ILO1 and ILO2. Instead, the understanding of the complexities and the fundamentals of software engineering were assessed indirectly by the terminology used by the students throughout the project and in their written deliverables.

Since the teams found it difficult to adjust their way of working to Scrum, their experiences of the Scrum practices were often superficial which led to imprecise descriptions of how they had implemented Scrum and what they learnt from their application. Instead, quite a few of the teams focused on technical descriptions of the technology they had used or the product they delivered. Furthermore, a recurring situation among those teams that did describe their process decisions focused on what happened but not alternative paths, turning the reflection into an experience report. This is also mirrored by the fact that only 3 out of 29 teams made a clear connection to the literature or the guest lectures when reflecting on their own praxis.

Among the comments given by the students in the course survey, we could see both that *“It’s not easy to divide the learning goals into concrete goals which I can check if I learned”* as well as that *“It is very easy to understand what I was supposed to learn, but the course did not make it easy to learn”*. Table 2 shows the student responses to selected relevant statements in the course evaluation survey.

## 5 A Toolkit for Reflective Practice

The artifact we developed to address the shortcomings of the course described in the previous section is a toolkit composed of different learning activities, assessment tasks, and professional practices applied by the teachers. These components are complemented by guidelines for a course structure as well as a model of reflective practice in an educational setting that provides a framework for the deployment and use of the different tools. We are going to discuss the different elements of the toolkit

in the following using the model of reflective practice as a starting point and structure for our explanations.

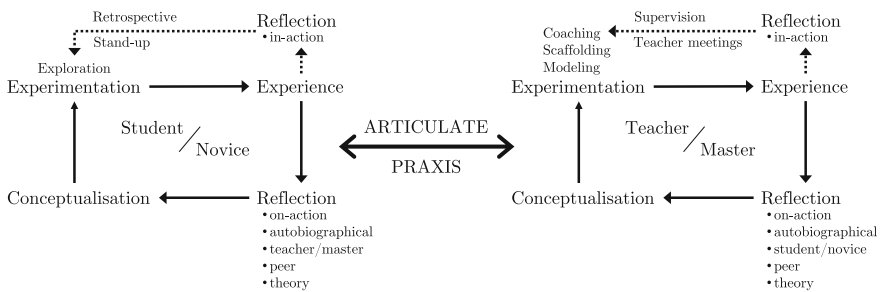
Our toolkit for reflective practice is not limited to teaching software processes. It is also appropriate for software architecture, testing, etc. since the ability to describe what is, what might or should be and how to bridge the gap is a useful exercise to include in all software engineering education. This is further elaborated for non-software engineering courses in Sect. 8.

### 5.1 Model of Reflective Practice

The model of reflective practice we developed (cf. Fig. 2) is based on reflection loops by both the student [who is the novice in terms of cognitive apprenticeship (Brown et al., 1989)] and the teacher (who is the master in terms of cognitive apprenticeship).

**Table 2** Mean and median student responses to questions in the course evaluation on a scale from 1 to 5 where 5 is best unless otherwise noted

Statement	Mean	Median
“I had enough prior knowledge to follow the course”	3.63	4
“The learning outcomes clearly describe what I was expected to learn in the course”	3.5	4
“The course structure is appropriate in order to reach the intended learning outcomes of the course”	2.54	2
“The teaching worked well”	2.53	3
“The assessment tested whether I had reached the intended learning outcomes of the course”	3.22	3
“The course administration worked well”	2.98	3
“The course workload as related to the number of credits was 1—too low, 5—too high”	3.31	3
“What is your overall impression of the course?”	2.69	3



**Fig. 2** A model of reflective practice, showing the different reflection loops for both teachers and students



ship). These reflection loops are connected to each other and follow the structure of Kolb's learning cycle (Kolb, 2014). Out of *experimentation* arises *experience* which is *reflected* upon either in-action or on-action (i.e., directly in the situation in which the experience is created or later on). This reflection leads to new insights that are either *conceptualised* before being used in a new round of experimentation or lead to new experimentation directly.

The *reflection-in-action loop* (Schön, 1983) shown at the top also operates on a different timescale than the lower one. For the students, the upper loop operates on the timescale of a sprint (usually about a week), while the lower loop operates on several weeks where the conceptualization is supported by deliverables in which the students document their reflection. For the teachers, the upper loop has a similar timescale to that of the students, but the lower loop operates on the scale of course instances, where conceptualization is performed after the end of each course instance and experimentation begins again with the new course. Thus, the student side represents a continuous learning process (Schön, 1983) facilitated by reflection whereas the teacher side represents in-course intervention and long-term course development.

There are numerous connections between these reflection loops. The teachers' conceptualization and its manifestation in experimentation provide the students with the opportunity to experience and enter their own reflection loop. On the other hand, the teachers observe the students' experiences and use them in their own reflection. More subtly, when teachers transition through Kolb's learning cycle from conceptualization to experimentation, they explain their own reasoning to the students and thus share their own reflections (see professional practices below) by *articulating* their *praxis* to the students. This is, in turn, a way to *model* the teachers' own reflections and serves as an example for the students in how they can reflect on their own experiences to better understand and form new concepts.

In summary, the model captures that reflection requires doing in that both students and teachers are involved in various actions and thereby also acquire shared experiences throughout the course. In addition, since reflection is conceptual it is not planned as a one-shot activity but repeatedly carried out during the course and different viewpoints are shared and considered to be able to form informed concepts (cf. Sect. 2).

## 5.2 Course Structure

The second part of the toolkit is the way the course is structured. In general, we try to get the students started as quickly as possible and avoid theory-laden lectures at the start. Instead, we apply the practical learning activities early on and then begin the iterative–incremental development quickly. The main part of the course is thus organized in sprints where each sprint starts with a planning session and concludes with a review and a retrospective. Students are encouraged early on to take other obligations into account when planning a sprint and set a reasonable velocity for each. Necessarily, estimations of velocity and of the user story effort are unreliable

in the beginning, but we encourage the students to learn from these mistakes and continuously improve their estimations based on their reflections. The reviews are coupled with a supervision learning activity, but are itself focused on the product and thus ideally conducted by a third party that provided the project. Students perform their retrospectives on their own, but need to record them (cf. Sect. 5.4).

### 5.3 Learning Activities

An important part of the toolkit is learning activities that provide students with shared experiences that they can use to develop their knowledge and skill through reflective practice. The learning activities are thus designed to trigger reflection in the students and are accompanied by specific assessment tasks that reinforce this (see below). In particular, we are utilizing three learning activities, further discussed below: a Lego Scrum simulation, a Kata for learning about scientific thinking and continuous improvement, and an exercise to teach students how to break down and estimate tasks.

To familiarize students with a modern software development process, we utilize *Lego Scrum simulations* (Steghöfer, Burden, Alahyari, & Haneberg, 2017). In these simulations, the students apply the Scrum methodology to build a Lego city based on user stories provided by the teachers. An essential element of the simulation is that the students need to reflect after each sprint and learn from their experiences. The setup of the simulations forces some issues—e.g., communication problems and a lack of planning—to come up that negatively affect the students. Through the reflective practice in the retrospectives, by reflections done by the product owner during the sprint reviews, and by reflection-on-action after the simulation, the students improve their process and approach during and after the simulation.

In order to help students understand the reflective cycle (cf. Fig. 1), we are using “*Kata to Grow*” (Rother, 2017), a simple exercise in which students apply repeated experiments to reach their goal based on the analysis of their current condition. The goal is to complete a jigsaw as quickly as possible and the students improve this process with changing constraints (e.g., “all jigsaw pieces need to be face down at the start”) iteratively based on “experiments” they devise and measurements of their current state. The kata, therefore, embodies a reflection loop and shows how small improvements based on clear measurements and a defined goal can make tremendous differences. This thinking is reinforced when students are later asked to define KPIs and reflect on them in their assessment.

Finally, we are using the *Elephant Carpaccio* (Kniberg & Cockburn, 2013) exercise to demonstrate how a large assignment can be accomplished by cutting it into very thin slices—like eating an elephant. The students are asked to create small implementation tasks for a shipping cost calculator that are prioritized to deliver customer value as quickly as possible. During the exercise, the students are asked to share their reasoning in multiple iterations. After each iteration, students reformulate

the tasks. The exercise concludes with a reflection on how the exercise went, what the students learnt, and how this relates to the upcoming project.

## 5.4 Assessment Tasks

A particular focus of the assessment tasks is to embed each individual learning activity into the overall learning process. For that purpose, supervision sessions and hand-ins are distributed over the duration of the course. The final deliverable is used for setting the grade and should contain the content of all the previous hand-ins. This allows us to spread the workload over the course, get input for our reflection in the classroom (see below), and incentivizes the students to reflect on the learning activities right after they took place. After the Lego Scrum simulation, e.g., the students are asked to reflect on their experience and how it influences the way they set up the development process for the project.

An important ongoing assessment strategy is the *process supervision* performed by the teachers with each of the groups on a weekly basis. These supervisions are coupled with the reviews the groups conduct with the Product Owner and precede the groups' own retrospectives. In the supervisions, no technical details of the solution are discussed. Instead, the students are asked to describe their experiences with the process and associated topics such as teamwork. If students have trouble formulating issues themselves, they are nudged along those lines by questions such as "which aspect required more time than you expected?" followed by asking about the lessons learned. This triggers a reflection process that allows the teams to analyze shortcomings in the process and to have a focused discussion later on in the retrospective. It also allows the teachers to provide an outside perspective on any challenges or plans for improvement.

Since the learning outcomes state that students should demonstrate the ability to "learn new tools and APIs on his/her own", the final deliverable should contain a reflection of how well these tools worked for them. The practices the students pick up (e.g., pair programming, continuous integration, or a certain merging scheme) then become part of the reflective practice again and students evaluate their own application and their usefulness in the context of their experience. This also generates ideas on how to apply these practices better in the future, allowing students to create connections to their praxis in coming courses and their professional careers, thus lifting the learning from the current course into a larger perspective.

Furthermore, students are asked to reflect on their overall process, including the sprint reviews and retrospectives, as well as the relationship between prototype, process and stakeholder value and the relationship between their process and the theoretical literature and guest lectures. These reflections are intended to let the students reflect on the purpose of a process and how its implementation influences its effectiveness. The process should be driven by stakeholder value, result in a prototype to deliver that value, and use the different activities in the Scrum lifecycle to evaluate both the quality of the product and the quality of the process. Since the guest lectures

illuminate process issues in a context that is different from the students', it allows the students to establish whether their experience is generalizable or not and how the industrial experience differs from their own.

In the final deliverable, students are also asked to reflect on the previous deliverables and how they influenced the progress of the team and the learning. This is intended as a kind of meta-reflection to let the students reflect on their own learning process. By revisiting previous decisions and their reasoning, students are able to see the impact of their choices and how they influenced their work.

## **5.5 Professional Practices**

Finally, the teachers themselves use reflective practice throughout the course, both in their own praxis as well as in front of the class. Reflections within the teacher group about the different course moments are in-action and allow teachers to react to emerging situations within the course. Regularly, the teachers also reflect in front of the class, thus providing an element of cognitive apprenticeships. For instance, at the beginning of a lecture, the teachers could discuss the past week or issues that have come up since they last saw the students. They would then discuss the current state, describe the state they would like to reach, and outline the plan they would like to take to get there. This kind of critical evaluation of the teachers' own work creates an atmosphere in which criticism is welcomed and students feel that their issues are being taken up. Finally, the teachers share their reflections on the course evaluation as well as on their own assessment as well as ideas and results of course development activities with the students.

# **6 The New Course Design**

The course instance to which we applied the full toolkit took place during the spring of 2017 and was taken by 50 undergraduate students with a major in Industrial Engineering and Management and 8 students from other undergraduate programs. We use the same structure to describe the course instance as in Sect. 4 to describe the new course design.

## **6.1 Intended Learning Outcomes**

Since the intended learning outcomes have not been identified as a prominent issue in the course, they have remained the same throughout our endeavor. Thus, the ILOs are the same as those found in Sect. 4.

## 6.2 Learning Activities

The learning activities are taken from our toolkit (cf. Sect. 5.3) and combined with professional practices (cf. Sect. 5.5). All activities align with our model (cf. Sect. 5.1). An overview of the course structure, how the learning activities and assessment tasks are distributed over the duration of the course, and how they relate to the model of reflective practice in Fig. 2 is given in Table 3.

The course as a whole is modeled after an iterative–incremental process that provides fast feedback to the students. A main principle to achieve this is to reduce the up-front theoretical lectures to a minimum and expose the students to practical experiences that they can use to reflect-in/on-action as quickly as possible. At the same time, these shared experiences are extremely useful in the classroom since the teachers can also reflect-on-action and help the students with the conceptualization of the experience. The other principle is to provide formative feedback to the students as often as possible in the second part of the course and to let them experience the whole reflective cycle (cf. Fig. 2) several times during the course.

The first lecture introduces the learning outcomes, the activities and the assessment tasks. We also describe what is new for this course instance based on the last course evaluation. This serves two purposes: first, it lets us communicate that we apply a reflective approach to our own course improvement; second, it lets us discuss what we came up with as concrete changes from reflecting on the course, thus employing one of our professional practices from the toolkit. As definition of reflection, we cite Smith’s “*assessment of what is in relation to what might or should be and includes feedback designed to reduce the gap*” (2001). We end the first lecture by introducing Scrum in terms of roles and activities in the lifecycle.

The next scheduled activity is the Lego Scrum simulation. In the simulation, the students carry out a mini project in terms of building a Lego city (Steghöfer et al., 2017). The students go through the reflective cycle several times (once in each of four sprints) with the aim to understand and improve how they conduct Scrum. Thus we practice an agile methodology in an iterative and incremental way, where each cycle builds on the previous one and includes explicit reflective activities. The students are then asked to reflect and conceptualize their findings in D1. As teachers, we treat each sprint as one iteration of the experiment—experience—reflection-on-action cycle. In each cycle, our reflection on how we perceive the student’s efforts influences our feedback to the teams during the review and the retrospective, what we want to accomplish with the next sprint, how the current exercise relates to previous exercises, and how we want to use the exercise next time around.

The practical activity of the Lego Scrum simulation is followed by a lecture where the more intricate details of Scrum as well as how to scale Scrum in a large organization is explained. The teachers relate the new theory to the experiences from the Scrum exercise and thus couple theory to practice and reflect on specific situations during the exercise together with the students.

Next up is the Kata to Grow exercise (Rother, 2017). The students are asked to complete a jigsaw as a team, iteratively improving their approach and reducing

**Table 3** The course structure, outlining the different activities. Elements taken from the toolkit or using elements of the toolkit are emphasized. The relation to the elements of the model in Fig. 2 is made from the teacher perspective (*T*:) and the student perspective (*S*:). Reflection-in-action and reflection-on-action are abbreviated as RiA and RoA, respectively

Week	Type	Activity/Task	Relation to model
0	Course preparation	Preparation of material, lectures, course plan, schedule, guest lectures, etc.	<i>T</i> : RoA, Conceptualization
1	Learning activity	<i>Lecture: Course introduction</i>	<i>T</i> : Experiment.; <i>S</i> : Concept.
		<i>Lego Scrum Simulation, Kata to Grow</i>	<i>T</i> : Experiment., RiA; <i>S</i> : Whole cycle
	Assessment	Technical supervision <i>D1: Reflections on Lego Scrum simulation</i> <i>D2: KPI</i>	<i>S</i> :RoA
2	Learning activity	Lectures: <i>Scrum &amp; Assessment, Software Quality</i>	<i>T</i> : Experiment., RiA; <i>S</i> : Concept., RiA
		<i>Elephant Carpaccio</i>	<i>T</i> : Experiment., RiA; <i>S</i> : Whole cycle
	Assessment	<i>Process supervision</i> <i>D3: Initial product backlog</i>	<i>T</i> : Experiment.; <i>S</i> : RoA, Concept.
3	Learning activity	Lecture: <i>Project background</i>	<i>S</i> : RoA, Concept., RiA
	Assessment	<i>Process supervision</i>	
4	Assessment	<i>Process supervision</i>	
5	Learning activity	<i>Guest lecture</i>	<i>S</i> : RoA, Concept., RiA
	Assessment	<i>Process supervision</i>	
		<i>D4: Half-time evaluation; reflection on the work so far</i>	
6	Learning activity	Guest lecture	
	Assessment	<i>Process supervision</i>	
7	Learning activity	<i>Lecture: Reflections on course and project</i>	<i>T</i> : RoA, Experiment.;
	Assessment	<i>Process supervision</i>	<i>S</i> : Concept.
8	Assessment	Final presentations <i>D5: Working prototype</i>	
9	Assessment	<i>D6: Reflection report</i>	<i>S</i> : RoA
10	Course Evaluation	Feedback from students and discussions in teacher group	<i>T</i> : RoA, Concept.; <i>S</i> : Concept.

the required time. In total, the students complete the plan-act-reflect cycle six times this way. We end the exercise by sharing our reflections on the outcome and introducing the concept of key performance indicators (KPIs). The exercise concludes with a presentation of KPIs other than time that can be used for evaluating process improvement.

The third exercise, Elephant Carpaccio (Kniberg & Cockburn, 2013), shows how a large assignment can be accomplished by cutting it into very thin slices. During the exercise, the students are asked to share their reasoning at multiple intervals and receive feedback for each iteration. The exercise concludes with a reflection on how the exercise went, what the students learnt and how it relates to the upcoming project.

We follow-up with more lectures on software engineering basics, such as requirements and testing and introduce the project topic. After that, the lecture format shifts to guests from industry presenting their experiences from agile software development where each presentation is limited to the first half of the lecture. The second half is then used to reflect on how the guests' experiences resonate with those of the students and how they tie into the learning outcomes of the course. The guest lectures are in this way a possibility for the students to reflect on the praxis of a professional in relation to their own experiences.

In the last lecture, we repeat how the course evaluation has led to changes to the current course instance, how we assess the outcome and what we propose to change for the next course instance, thus highlighting our own plan-act-reflect cycle. The students are also given the opportunity to share their reflections on how the course panned out. The ILOs are then discussed with the students and they are asked to reflect on which opportunities they have had to reach the ILOs and what kind of assessment they have been given or expect to receive. The last lecture concludes by detailing the remaining deliveries and how the final presentation will be handled.

### 6.3 *Assessment*

The assessment tasks are now both formative and summative. We use the supervision slots to give the students formative feedback on their application of Scrum and discuss other aspects of the course as the teams find appropriate for their current needs. This is also inspired by agile practices: we aim to provide the students with fast and frequent feedback to adapt their behavior as they go along, instead of having to rely on a single feedback opportunity at the end of the course. The supervisions are divided into feedback on the product the students are building (modeled after sprint reviews) and feedback about the process the students apply (modeled after sprint retrospectives). The teachers are only engaged in the latter kind of feedback while the sprint review is conducted with an external Product Owner.

Assessment is also done by the teams handing in six different deliverables during the course, most of which contain elements of reflection as described in Sect. 5.4:

**D1:** Three reflections from the Lego exercise in terms of what the team would like to continue doing, stop doing or do differently when they apply Scrum in their project. The changes should be motivated and feasible to implement. D1 serves as the basis for a session where we select some of the reflections to illustrate how the assessment strategy will be applied. The teams also submit a social contract

detailing their ambition levels, when and how to have meetings, etc. D1 is handed in at the end of the first-course week.

- D2:** After the Kata exercise, the teams are asked to choose three KPIs to monitor the strategies detailed in D1. To be handed in by the end of week two.
- D3:** When the project scope has been introduced, the teams are asked to come up with an initial product backlog in week three. The backlog can contain epics and larger elements but should have enough user stories to fit the first sprint. D3 is then used during the Elephant Carpaccio exercises to illustrate how large stories can be split more and more thinly.
- D4:** Half-way through the project, in week six of the course, the teams are asked to hand in a one-page document reflecting on the work so far, both in terms of process and product. At the subsequent supervision slot, the teams pair up to facilitate the sharing of experiences and insights across teams but also to give opportunities for reflecting on each other's progress.
- D5:** The fifth deliverable is a working prototype for the final presentation in week eight. It does not need to be documented but it should be executable so that the students can demonstrate how they have chosen to tackle the project scope and what value they deliver to the PO.
- D6:** The last deliverable consists of the source code and the output from a git repository analysis tool as well as the artifacts asked for under Prototype and the Reflection Report (see below). D6 is handed in at the end of week nine which is the last week of the course.

The final team grade now relies on three elements:

**Value** The relevance and completeness of what is delivered in relation to how the teams have defined the scope of the project based on what the stakeholder has asked for makes up 24% of the final grade.

**Prototype** 30% of the final grade is based on the documentation, automatic code quality analysis, automatic and manual tests as well as design decisions.

**Process** Reflecting on how the team has applied Scrum as well as on the intermediate deliveries D1 to D5, describing their best practices for using new tools, and how their process relates to literature and guest lectures make up the remaining 46% of the final grade.

Just as in the old course design, we use personal evaluations and metrics from the code base to assess if there are team members that deserve a higher or a lower grade than the team grade. Deviations are never based on one source but need to be anchored in both and are often supplemented by our own observations during supervision or follow-up discussions.

## 6.4 *Constructive Alignment and Student Perception*

We supply data to show what the students report w.r.t. the course design in Table 4. Since both what we assess and how we prepare the students is different, it is not



**Table 4** Mean and median student responses to questions in the course evaluation for Spring 2017 on a scale from 1 to 5 where 5 is best unless otherwise noted

Statement	Mean	Median
<i>“I had enough prior knowledge to follow the course”</i>	3.85	4
<i>“The learning outcomes clearly describe what I was expected to learn in the course”</i>	4.35	5
<i>“The course structure is appropriate in order to reach the intended learning outcomes of the course”</i>	4.25	4
<i>“The teaching worked well”</i>	4.45	5
<i>“The assessment tested whether I had reached the intended learning outcomes of the course”</i>	4.10	4
<i>“The course administration worked well”</i>	4.25	5
<i>“The course workload as related to the number of credits was 1—too low, 5—too high”</i>	3.30	3
<i>“What is your overall impression of the course?”</i>	4.20	5

possible to say what has caused the change in student perception. However, the overall increase of the scores indicates that the new course design is not seen as contradicting a good learning situation as well as supporting our view as teachers that the new design supports the students’ ability to reach the intended learning outcomes.

In response to the free text question about what should be kept for the next course instance, one student replied *“The practical setup and “trial and error” approach. You learn better from making mistakes, rather than doing it right the first time.”* This implies that we still have something to work on. Not only mistakes should drive learning, but reflecting on the practical experience as a whole, including both problems that occurred and things that went well. However, mistakes tend to force reflection since the identification of the mistake resonates with the description of what is. If the mistake is to be corrected, a change is needed which also encourages to consider what should be and feedback to reduce the gap. But success is also an experience worth reflecting over since understanding what enabled the success and how it can be repeated saves both effort and time in the future.

In terms of constructive alignment, comparing Table 1 with Table 5 shows that the intended learning outcomes are now addressed with additional exercises, thus emphasizing skill development and practical experience instead of a mostly theoretical approach. Since these exercises are always connected to reflections, this element is significantly strengthened accordingly.

In relation to ILO7—students should be able to learn new tools by themselves—the lectures and exercises do not mention how to use the new tools but reflect on what the students have experienced during the exercises and how that can be transferred to the project. The exercises also present teachers and students with shared experiences that can serve as basis for reflecting together as illustrative examples to explain concepts and strategies for handling these. For instance, the students played with

**Table 5** The course alignment matrix for the new course design

ILO	Learning activity	Assessment tasks
ILO1	6 lectures, 2 exercises and project	5 supervisions, D2 and terminology
ILO2	6 lectures, 2 exercises and project	5 supervisions and terminology
ILO3	6 lectures, 2 exercises and project	5 supervisions, process and terminology
ILO4	4 lectures, 2 exercises and project	5 supervisions, D3 and prototype
ILO5	3 lectures, 3 exercises and project	5 supervisions, D1, D2, prototype and process
ILO6	3 lectures, 3 exercises and project	5 supervisions
ILO7	1 lecture, 3 exercises and project	5 supervisions and process
ILO8	4 lectures, 3 exercises and project	5 supervisions, D1, D4 and process

Lego as kids but still struggle with finding the right Lego pieces for their buildings. This shared experience is something we can go back to as we reflect on how their programming skills might transfer to using a new API and development tools.

## 7 Reflections on the Toolkit

This section will first detail how the students utilized the opportunities for reflection-in- and -on-action, before we describe our own thoughts and relate those to existing literature.

### 7.1 Student Lens

Immediate reflection-in-action is relevant as events unravel during the course and students need to handle situations for which they are not prepared. These reflections are sometimes difficult to document due to the time and place when and where they occur. However, the students have recurring opportunities to reflect on their experiences, e.g., during daily stand-up meetings or sprint retrospectives. These opportunities are easier to document and reflect on since they occur at defined points during the sprints and allow to define what should be and how to bridge the gap while the project still runs. The teams' reflection-on-action is documented in the reflection report after the final presentation (D6), meaning that the students do not have the possibility to implement their suggested feedback within the course. Instead, the intention is that the insights will be of use in their future studies and professional life.

*Reflection-in-Action* Two teams decided to structure their reports to mirror Smith's definition of reflection by first describing what they did, and then described what they would do differently and how. The first team consistently used the subheadings "*The*

*situation as it is*” and “*What we would like it to be*” where the latter also included strategies for realizing the change. The other team defined what went well, what could improve and how they could improve for each of the bullets required for the reflection report. As an example, they stated that their communication with the PO went well, that they could improve in how they used roles within the team and that the improvement could be realized by not only assigning responsibilities but also defining what each responsibility covered.

One of the student teams wrote in their reflection report that they would include how to conduct daily Scrum meetings in their social contract. Since the team members took different parallel courses, they had difficulties finding a time that suited everyone. They therefore suggested to regulate how all team members can participate even if they cannot be physically present upfront. Another team stated that it was difficult in the beginning to keep the meetings short and concise since when a team member described what they had done, other members wanted to know how a specific task had been solved. The meetings, therefore, tended to involve lengthy technical descriptions. The team came up with two strategies to shorten the meeting time. First, they decided to stand up during the meetings since this improved focus and was recommended by literature. Second, they planned meetings where insights regarding how to handle new tools and technologies could be shared. Their conclusion was that while it is important to share information it is also important to know when to share what.

A similar experience was reported by a third team in relation to the sprint retrospectives. In the beginning of the project, these were held at the supervision slot and as a consequence just after the retrospective and before the planning as well as in the same location. The discussions quickly became technical and focus shifted from process to product. Therefore, they decided to have the retrospectives at another physical location and ban visible computers. In this way, the focus on process improved and the team reports that their satisfaction with the retrospectives increased over time.

*Reflection-on-Action* Regarding the peer lens, one team stated that it was helpful to see how another team handled the same challenges they faced. However, they did not provide details about the challenges and what they could have done differently. Other teams were more articulate but concluded that the peer discussion came a week too late for them to have a real impact. By the time they were asked to reflect on their first two deliverables, they had just overcome a major obstacle in how to communicate with the shared backend. Therefore, they felt that the rest of the sprints would be more straightforward and would allow the team to focus on delivering value instead of debugging. This gave them the opportunity to assess what lay ahead and to evaluate what they just had done in relation to what they thought they would do.

In relation to the first delivery (D1), one team felt that they were initially right in stating the importance of understanding the needs of the product owner (PO) instead of the desired solution since there might be other ways of delivering value: “*Focus was on how our sketch and vision could be adapted to the PO’s instead of understanding why the PO came with a specific solution*”. Half-way through the project the team managed to shift focus and concentrate on the context of the PO and

from there redirect their development effort towards a system more suitable for the needs of the PO.

An example of how a team identified their own learning progress throughout the project relates to the definition of done that they used for their user stories: “*As our understanding of the system and project grew, it became easier to identify and structure these definitions.*” As we saw in relation to daily Scrums and the introduction of meetings with the specific purpose of sharing knowledge between team members, the peer lens was also applied within teams to share knowledge and reflect on how to improve their way of working.

An interesting observation is that while all teams relate their reflections on the course literature and the guest lectures, none of the teams relate their reflections to what the teachers have said.

## 7.2 *Teacher Lens*

The toolkit for reflective practice proposed in Sect. 5 allowed us to address the gap between what we imagined the course to be and what it was. It is the result of a 3-year effort to improve the course and move it from a product-focused programming project with poor constructive alignment and a mismatch between theory from lectures and what was practically applied towards a process-focused engineering course that is driven by practical experience and continuous reflection.

*Reflection-on-Action* Our own perception of the course has improved significantly with the introduction of the different elements of the toolkit for reflective practice. While the course is still known amongst the student body as “the android course”, we are confident that we now focus on the process issues that is at the heart of the intended learning outcomes much better. This also makes it easier for us to communicate our vision for the course to the students. The expectations of the students and the place of the course in the different programs are also much clearer. Instead of being yet another development project, the course now offers different and novel content.

In relation to the old course design, the project now starts on the third week of the course. Instead of letting the students immediately get to work on the project we use the first weeks to introduce the central concepts and Scrum. These concepts are then explored during three exercises where each exercise has a component of reflection and feedback. This change means that there are fewer supervisions but also that the students get help in bridging the gap between theory presented in the lectures and the practice they are asked to explore during the project as well as an opportunity to reflect on what they have done during the exercise and what they want to do during the project.

Including collective feedback into the lectures also means that we as teachers not only have the opportunity to motivate the exercise and the deliverable, we can also reflect on what went well with the exercise and how we aim to improve it for the next time. We thereby verbalize our own reflection in front of the students and

model how we came to give the exercise the way we did. In this way, the exercises supply a scaffold for the students to reflect on how they plan, execute, and evaluate a team project as well as agile practices like splitting user stories into tasks. Since supervision is handled by the teachers with a deliberate focus on process matters they provide opportunities to coach the students in their reflective practice based on the ideas, uncertainties, and milestones they want to bring up. We as teachers can also bring up topics we find worth discussing. Throughout the different activities, we can go back and relate what is happening and how we reflect within the current context to the shared experiences we obtained through the three exercises. In this way, our new course design mirrors the recommendations to combine subject matter with reflective practice so that the task becomes more concrete and has an immediate bearing on the students' professional development (Mathiassen & Pura, 2002).

An important aspect of this new structure and the progression of assessment tasks is that we are able to build a trustful relationship with the students (Gunn, 2010). Since they have the opportunity to receive formative feedback continuously but only the final hand-in is graded, they understand our expectations and how they can address them much better. Trust is also built by articulating our own reflections and being open about problems in the course and how we are going to address them. We thus demonstrate that failure is an opportunity to learn and that admitting mistakes is an important step in the learning process. We thus allow a cognitive apprenticeship to form in the classroom.

*Reflection-in-Action* Having reflected on what we do and how we want to improve the course gives us an understanding of what we want to achieve with the different learning activities and the corresponding assessment. At the same time, we also gain new experiences each time we give the course. By sharing these experiences and how we acted and reasoned provides us teachers with a portfolio of situated reflections that we can rely upon when we encounter situations for which we are not prepared. In this way, reflection-on-action supports our reflection-in-action.

### **7.3 Theoretical Lens**

When comparing our own work with related literature, it becomes evident that reflective practice is a recurrent theme in software engineering education. In the work of Hazzan, e.g., reflection is seen as a driving factor in education about human factors in software engineering (2004). The same author also suggests to use reflection with a tutor as a way to continuously drive a project forward in a studio environment (Hazzan, 2002). However, Hazzan couples reflective practice in software engineering education directly to the specific method of the studio in which students meet with a capable tutor several times a week, thus increasing commitment and motivation and exposing the students to constructive criticism and different social interactions connected to collaborative work. While this method is very intriguing, it also requires significant resources, both in terms of meeting space and effort by the tutors. Such a

method is thus not feasible in the resource-constrained environment we find ourselves in.

Another take on the studio as an instrument for reflection is presented by Bull and Whittle (2014). They argue that project-based courses are better for facilitating reflection than lecture-based courses since they give students the opportunity to work iteratively. Still, such courses often suffer from considering reflection as an implicit learning objective and do not explicitly address it through the teaching activities. The authors conclude that the studio approach is recommendable for fostering reflection at program level and allows addressing learning objectives over multiple courses. We agree, while we also believe that our own course is an example of how a single course can introduce explicit learning objectives, activities, and assessment strategies that foster student reflection.

The studio method championed by Hazzan and Bull and Whittle is one example of the more abstract concept of *communities of practice* (Wenger, 1998). They regard learning as a social and collaborative effort that is based on the common passion for a subject and the interaction between the learners and the teachers. Our toolkit for reflective practice helps us in establishing such a community of practice: joint activities in the course within a common domain create a community that is based on practical experience and reflective practice about this experience. Continuous interaction between the students and between students and teachers and the learning activities are designed to create a “shared repertoire of resources” that helps the students in their learning process and in achieving the intended learning outcomes.

Reflection has also been acknowledged as a problem-solving strategy in software engineering education. For instance, teaching students how to reflect in order to improve their skills in writing software tests (Edwards, 2004) enables them to move away from a trial-and-error approach and thus allows them to find solutions more quickly and efficiently. In particular, the role of feedback for the success of reflective practice is emphasized in Edwards (2004). While this feedback is provided by an automated system in the course the contribution reports on, we aim to provide formative feedback in our supervision sessions with the students and in the different learning activities throughout the course.

In terms of assessment, the assessment strategies in our toolkit for reflective practice are instantiated, among others, in the final deliverable that contains a reflection report in which students reflect on their experience with the process. This is similar to the use of postmortem reports to evaluate software architecture projects suggested by Wang and Stalhane (2005). Such reports are often used in the industry to analyze a product development effort and draw conclusions that can support a software process improvement initiative (Dingsøy, 2005). However, the cited paper proposes to only include positive and negative experiences in the report. While this is an important part of reflective practice, the crucial part of deriving concrete improvement steps and evaluating those in practice—an essential part of our toolkit for reflective practice—is missing.

## 8 Applying the Toolkit Outside SE Education

While we developed the toolkit for use in a software engineering project course, its general outline should be applicable to different course structures within engineering and the sciences in general. The model of reflective practice (cf. Sect. 5.1) is independent of concrete course content and only mandates an iterative approach. The assessment tasks (cf. Sect. 5.4) we use are likewise independent of the concrete product or discipline of engineering and focus on reflecting on the students' praxis and choices. Similarly, the professional practice of the teachers (cf. Sect. 5.5) of reflecting about the course amongst themselves and in front of the class is completely independent of the concrete discipline being taught.

We see the main application area in engineering project courses in which an artifact needs to be developed by students following a specific process. In these situations, students often exhibit a "product over process" attitude (Steghöfer et al., 2016). Using reflective practice that is focused on the process draws the students' attention to these issues and makes it easier for the teachers to put process aspects into the foreground. The course structure (cf. Sect. 5.2) is applicable in such project courses with minimal modifications based on the background knowledge and skills students need to acquire before being able to start working on the product and the length of the course. The learning activities (cf. Sect. 5.3) might also be adapted. While the kata and the process supervision are transferrable to other disciplines, the Lego Scrum exercise uses a dedicated *software* development process. However, other simulations or serious games could be used to achieve a similar effect. One example is the urban planning game described in Mayer, Carton, de Jong, Leijten, and Dammers (2004) in which students simulate the development of a city and the necessary negotiations between the involved stakeholders.

## 9 Conclusions

In this chapter, we have described the toolkit for reflective practice, a set of teaching, assessment, and professional practices based on a model of reflective practice for engineering courses with a particular focus on software engineering. We have shown how the toolkit was developed using action design research based on issues observed in a project course we teach and how the toolkit is applied in the current version of the course. The toolkit is thus an answer to RQ1: *How can we facilitate reflective practice in a software engineering project course?* Our discussion of the student perception, our own perception and the relation to other published work also shows that the toolkit is viable, thus providing an answer to RQ2: *How do students utilize opportunities for reflective practice for their continuous learning?*

It is important to note that our toolkit for reflective practice contains many aspects that allow us to teach how reflection actually works. Being able to reflect is a skill that needs to be acquired by our students. Our experience shows that students are

successful in doing this by following the teacher's example and by being encouraged to reflect continuously while the course is running.

In the future, we would like to make reflection of both students and teachers an even more prominent feature of the course. One way to achieve this is to instate daily stand-up meetings, a practice that many student groups already take up on their own. Another would be to start each lecture could with a reflection by the teachers. At the moment, this only happens if there are events that make it prudent to do so. A further possibility is to include additional opportunities for peer assessment in the course, where students perform peer reviews of the reflection reports of the other students to see positive and negative examples. Notably, being able to write a good review is another skill that we cannot expect from our students. Thus, reviewing would have to be introduced and formative feedback on the reviews would be necessary. However, since architecture and code reviews are common practices in software engineering, this could provide an additional opportunity to include an important professional practice in the course.

## References

- Anderson, L., Krathwohl, D., & Bloom, B. (2001). *A taxonomy for learning, teaching, and assessing: A revision of Bloom's taxonomy of educational objectives*. Longman.
- Babb, J., Hoda, R., & Nørbjerg, J. (2014, July). Embedding reflection and learning into agile software development. *IEEE Software*, 31(4), 51–57. <https://doi.org/10.1109/MS.2014.54>.
- Basili, V. R. (1993). The experience factory and its relationship to other improvement paradigms. In *European Software Engineering Conference* (pp. 68–83). Springer.
- Basili, V. R., & Caldiera, G. (1995). Improve software quality by reusing knowledge and experience. *MIT Sloan Management Review*, 37(1), 55.
- Biggs, J. (1996). Enhancing teaching through constructive alignment. *Higher Education*, 32(3), 347–364.
- Bourque, P., Fairley, R. E., et al. (2014). *Guide to the software engineering body of knowledge (SWEBOK (R)): Version 3.0*. IEEE Computer Society Press.
- Brookfield, S. (1995). *Becoming a critically reflective teacher*. San Francisco: Jossey-Bass.
- Brown, J. S., Collins, A., & Duguid, P. (1989). Situated cognition and the culture of learning. *Educational Researcher*, 18(1), 32–42.
- Bull, C., & Whittle, J. (2014, July). Supporting reflective practice in software engineering education through a studio-based approach. *IEEE Software*, 31(4), 44–50.
- Burden, H. (2017). DAT255 Software Engineering Project, HT2014. Retrieved March 29, 2018, from <https://github.com/hburden/DAT255/tree/ht2014>.
- CMMI Product Team. (2010). CMMI for development, version 1.3 (tech. rep. No. CMU/SEI-2010-TR-033). Software Engineering Institute, Carnegie Mellon University.
- Dingsøyr, T. (2005). Postmortem reviews: Purpose and approaches in software engineering. *Information and Software Technology*, 47(5), 293–303.
- Dingsøyr, T., Moe, N., Schalken, J., & Stålhane, T. (2007). Organizational learning through project postmortem reviews—An explorative case study. *Software Process Improvement*, 136–147.
- Dresch, A., Lacerda, D. P., & Antunes, J. A. V. (2014). *Design science research: A method for science and technology advancement*. Springer Publishing Company, Incorporated.
- Edström, K. (2008). Doing course evaluation as if learning matters most. *Higher Education Research & Development*, 27(2), 95–106. <https://doi.org/10.1080/07294360701805234>. eprint: <http://dx.doi.org/10.1080/07294360701805234>.



- Edwards, S. H. (2004). Using software testing to move students from trial-and error to reflection-in-action. In *Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education, SIGCSE '04* (pp. 26–30). Norfolk, Virginia, USA: ACM. <https://doi.org/10.1145/971300.971312>.
- Elmgren, M., & Henriksson, A. (2010). *Universitetspedagogik*. Norstedts.
- Freire, P. (2000). *Pedagogy of the oppressed: 30th anniversary edition*. Bloomsbury Academic.
- Gunn, C. L. (2010). Exploring MATESOL student ‘resistance’ to reflection. *Language Teaching Research, 14*(2), 208–223.
- Hazzan, O. (2002). The reflective practitioner perspective in software engineering education. *Journal of Systems and Software, 63*(3), 161–171.
- Hazzan, O., & Tomayko, J. E. (2004, March). Reflection processes in the teaching and learning of human aspects of software engineering. In *17th Conference on Software Engineering Education And Training, 2004. Proceedings* (pp. 32–38). <https://doi.org/10.1109/CSEE.2004.1276507>.
- Huber, G. P. (1996). Organizational learning: A guide for executives in technology-critical organizations. *International Journal of Technology Management, 11*(7–8), 821–832.
- Jørgensen, M., & Gruschke, T. M. (2009, May). The impact of lessons-learned sessions on effort estimation and uncertainty assessments. *IEEE Transactions on Software Engineering, 35*(3), 368–383. <https://doi.org/10.1109/TSE.2009.2>.
- Kniberg, H., & Cockburn, A. (2013). Elephant Carapaccio exercise. Retrieved October 30, 2017, from <https://docs.google.com/document/d/1TCuuu8Mm14oxsOnlk8DqfZAA1cvtYu9WGV67YjsSk/pub>.
- Kolb, D. A. (2014). *Experiential learning: Experience as the source of learning and development* (2nd ed.). FT Press.
- Loughran, J. J. (2002). Effective reflective practice in search of meaning in learning about teaching. *Journal of Teacher Education, 53*(1), 33–43.
- Lyons, N. (Ed.). (2010). *Handbook of reflection and reflective inquiry—Mapping a way of knowing for professional reflective inquiry*. New York, NY: Springer.
- Maham, M. (2008, August). Planning and facilitating release retrospectives. In *Agile 2008 Conference* (pp. 176–180). <https://doi.org/10.1109/Agile.2008.60>.
- Mann, S. (2005, July). The language teacher’s development. *Language Teaching, 38*, 103–118.
- Mathiassen, L., & Purao, S. (2002). Educating reflective systems developers. *Information Systems Journal, 12*(2), 81–102.
- Mayer, I. S., Carton, L., de Jong, M., Leijten, M., & Dammers, E. (2004). Gaming the future of an urban network. *Futures, 36*(3), 311–333.
- Petterson, F., Ivarsson, M., Gorschek, T., & öhman, P. (2008, June). A practitioner’s guide to light weight software process assessment and improvement planning. *Journal of Systems and Software, 81*(6), 972–995. <https://doi.org/10.1016/j.jss.2007.08.032>.
- Rother, M. (2017). Kata to grow—A simple, free exercise to help teach scientific thinking [online]. Retrieved October 30, 2017, from <https://www.katatogrow.com/>.
- Schön, D. A. (1983). *The reflective practitioner: How professionals think in action*. Harper torch-books. Basic Books.
- Sein, M. K., Henfridsson, O., Purao, S., Rossi, M., & Lindgren, R. (2011). Action design research. *MIS Quarterly, 35*(1), 37–56. Retrieved from <http://www.jstor.org/stable/23043488>.
- Shkedi, A. (2000). Educating reflective teachers for teaching culturally valued subjects: Evaluation of a teacher-training project. *Evaluation & Research in Education, 14*(2), 94–110.
- Smith, R. A. (2001). Formative evaluation and the scholarship of teaching and learning. *New Directions for Teaching and Learning, 2001*(88), 51–62.
- Steghöfer, J.-P., Burden, H., Alahyari, H., & Haneberg, D. (2017). No silver brick: Opportunities and limitations of teaching Scrum with Lego workshops. *Journal of Systems and Software, 131*, 230–247.
- Steghöfer, J.-P., Knauss, E., Alégroth, E., Hammouda, I., Burden, H., & Ericsson, M. (2016, May). Teaching agile—Addressing the Conflict between project delivery and application of agile. In

*Software Engineering Education and Training Track, the 38th International Conference on Software Engineering, Austin, TX.*

Tomal, D. R. (2010). *Action research for educators*. Rowman & Littlefield Publishers.

Turns, J., Sattler, B., Yasuhara, K., Borgford-Parnell, J., & Atman, C. J. (2014). Integrating reflection into engineering education. In *Proceedings of the ASEE Annual Conference and Exposition*. ACM.

Villalón, J. A. C.-M., Agustín, G. C., Gilabert, T. S. F., Seco, A. D. A., Sánchez, L. G., & Cota, M. P. (2002). Experiences in the application of software process improvement in SMES. *Software Quality Journal*, 10(3), 261–273.

Wang, A. I., & Stalhane, T. (2005, April). Using post mortem analysis to evaluate software architecture student projects. In *18th Conference on Software Engineering Education Training (CSEET'05)* (pp. 43–50). <https://doi.org/10.1109/CSEET.2005.42>.

Wenger, E. (1998). *Communities of practice: Learning, meaning, and identity*. Cambridge University Press.

Williams, L., & Cockburn, A. (2003, June). Agile software development: It's about feedback and change. *Computer*, 36(6), 39–43. <https://doi.org/10.1109/MC.2003.1204373>.

# Lean Learning of Risks in Students' Agile Teams



Wentao Wang, Chaitra Thota, Xiaoyu Jin, Nan Niu and Carla C. Purdy

**Abstract** Risk—the possibility of resulting in an unsatisfactory outcome—is an important driving force for a software development project to progress. Although techniques like identifying a project's top-10 risk items are taught commonly in software engineering courses, little work has been carried out to examine how students working in agile teams perceive and mitigate the risks over multiple software development cycles. In this chapter, we summarize our recent work where we discovered the collaborative nature of students' risk management strategies. Furthermore, we show that students also followed lean practices by wasting little effort on non-actionable risks. Linking collaboration and waste-elimination provided additional insights into teaching a wider range of lean principles in agile settings, e.g., students should deliver as fast as possible the non-collaborative risk mitigations but should decide as late as possible when facing interdependent mitigations.

**Keywords** Risks · Risk management · Agile software development · Agile teams  
Lean learning

---

W. Wang · C. Thota · X. Jin · N. Niu (✉) · C. C. Purdy  
Department of EECS, University of Cincinnati, Cincinnati, OH 45221, USA  
e-mail: [nan.niu@uc.edu](mailto:nan.niu@uc.edu)

W. Wang  
e-mail: [wang2wt@mail.uc.edu](mailto:wang2wt@mail.uc.edu)

C. Thota  
e-mail: [thotava@mail.uc.edu](mailto:thotava@mail.uc.edu)

X. Jin  
e-mail: [jinxu@mail.uc.edu](mailto:jinxu@mail.uc.edu)

C. C. Purdy  
e-mail: [carla.purdy@uc.edu](mailto:carla.purdy@uc.edu)

© Springer Nature Singapore Pte Ltd. 2019  
D. Parsons and K. MacCallum (eds.), *Agile and Lean Concepts for Teaching and Learning*, [https://doi.org/10.1007/978-981-13-2751-3\\_13](https://doi.org/10.1007/978-981-13-2751-3_13)

# 1 Introduction

A software project can face many risks throughout its lifecycle, from construction to deployment and maintenance. Risk is defined as any potential situation or event that negatively affects the project's success (Ropponen & Lyytinen, 2000), and such failure can be multifaceted: integration difficulty (Kamble, Jin, Niu, & Simon, 2017), requirements dissatisfaction (Niu & Easterbrook, 2007), lack of reuse (Niu, Bhowmik, Liu, & Niu, 2014a; Niu, Savolainen, Niu, Jin, & Cheng, 2014b), and so on. Risk itself is neutral, but if ignored or unmanaged, it can lead to project failure.

Risks, then, serve as an essential vehicle for a class of iterative and incremental process models, e.g., the spiral model (Boehm, 1986) and the agile software development (Beck et al., 2001). Agile incorporates the rapid and evolutionary development styles, and is now ubiquitous in the industry (Niu, Brinkkemper, Franch, Partanen, & Savolainen, 2018). Therefore, before the students enter the software industry, it is important for them to learn and practice agile software project development. Many teaching and education experiences in agile development have been reported (Schroeder, Klarl, Mayer, & Kroiss, 2012; Rico & Sayani, 2009; Anslow & Maurer, 2015; Devedzic & Milenkovic, 2011), but little is known about students' own perceptions of risks. Having this knowledge gap is crucial because risk is fundamental to change (Boehm, 1986; Bhowmik et al., 2016) and the agile manifesto values prompt responses to change (Beck et al., 2001).

Our recent work (Thota, Niu, Wang, & Purdy, 2017) examined the risks that students identified and mitigated in their agile projects while learning software engineering. In this chapter, we further analyze those risks as they relate to lean principles. In particular, we show that students' lean learning is reflected by the trend of their working on a smaller number of non-actionable risks. Linking this waste-elimination pattern and the collaborative risk management emerging from our recent study (Thota et al., 2017) offered additional insights into teaching other lean principles in agile settings. In our work, risks serve as the underlying connections between agile and lean. Specifically, mitigating identified risks is viewed as a key driver for agile teams to organize themselves and reducing the effort in non-actionable risks is considered to reflect students' lean learnings.

This chapter is organized as follows. Section 2 surveys the literature on risk management and its education in agile and lean contexts. Section 3 introduces our software engineering classes from which the risks of students' agile teams are collected. Section 4 presents the results where the students' risks are analyzed, especially from the collaborative and lean dimensions and their interactions. Section 5 discusses our work's limitations and implications. Section 6 concludes the chapter.

## 2 Background and Related Work

Risk management is fundamental for industrial software development, and thus often taught in software engineering classes. Risk management is about managing risk items that refer to some particular aspects of software development task, environment, or process which, if ignored, will increase the likelihood of a project failure (Ropponen & Lyytinen, 2000). Managing can be decomposed into risk identification, analysis, mitigation, and monitoring. There are some techniques commonly taught in (undergraduate) software engineering courses, such as calculating risk exposure and identifying and monitoring top- $n$  risk items (e.g.,  $n = 10$ ) (Boehm, 1991; Reifer, 2002; Boehm, 2007; Koolmanojwong & Boehm, 2013). In agile projects, one may encounter risk items like intrinsic schedule flaws and productivity variation (Cohn, 2013). Despite the perceived risks from client or contractor perspectives (Schmidt, Dart, Johnston, Sterling, & Thorne, 1999), the focus of our chapter is from the learning and practice perspectives of students themselves.

When teaching a two-semester sequence of a graduate-level software engineering course where the students worked in a five- or six-people team, Koolmanojwong and Boehm (2013) reported that better risk management was correlated with better grades. Furthermore, it was observed that students who identified risks in an earlier semester would mitigate those risks collaboratively in the later semester. To help organize risks, standards and reference models such as ISO 31000 (ISO 2018) and the Software Engineering Institute's potential risk items (Carr, Konda, Monarch, Ulrich, & Walker, 1993) have been developed. When exposed to the 194 questions aimed for a comprehensive identification of risks, the students perceived only 36 questions relevant in the waterfall project development (Collofello & Pinkerton, 1997).

In contrast to practicing waterfall software development, agile methodologies such as Scrum and XP (extreme programming) have been taught in an increasing number of courses (Rico & Sayani, 2009; Devedzic & Milenkovic, 2011; Schroeder et al., 2012; Anslow & Maurer, 2015). One subculture within the agile community advocates the transformation of lean manufacturing principles and practices to software development (Poppendieck & Poppendieck, 2003). Lean manufacturing (often simply "lean") refers to a systematic method for waste minimization so that the right products are made for the right customers (both internal and external) at the right time and in the right amount to achieve perfect work flow (Holweg, 2007). The successful applications in service industries (Hanna, 2007), business management (Radnor, Walley, Stephens, & Bucci, 2006), healthcare (Ker, Wang, Hajli, Song, & Ker, 2014), and other domains have positioned lean a solid conceptual framework as well as a practical tool suite. Poppendieck and Poppendieck (2003) described 22 tools and compared them to different agile practices. As a result, agile organizations are empowered by such lean principles as "eliminate waste", "see the whole", "build integrity in", "decide as late as possible", "deliver as fast as possible", and the like (Poppendieck & Poppendieck, 2003). Examining the interactions of risks and these lean principles in students' agile projects is a focus of our work.

Previous work by Emiliani (2004) attempted to introduce the lean principles in educational settings. Emiliani (2004) applied lean principles to the design and delivery of a graduate business course on leadership taken by part-time working professional students. The principles instructed were centered on “eliminate waste” and were instantiated by various practices such as direct and simplified readings, clear and targeted assignments, a summary of content in visual control, and so forth. The results indicated a higher level of student satisfaction (Emiliani, 2004).

Similar to Emiliani’s work, in this chapter, we have focused on the core lean principle of “eliminate waste”. However, in this chapter, we are analyzing undergraduate students’ perceptions of their agile projects’ risks and shed light on some other lean principles like “decide as late as possible” and “deliver as fast as possible” in software engineering teaching and learning.

### 3 Software Engineering Courses

This section introduces the software engineering courses in which the undergraduate students were grouped in agile teams to meet project milestones. In addition to the working software, the students were required to deliver the risks as part of their agile reflection documents at each milestone.

For the Spring 2015 semester, we chose iTrust as the working system for our students to engineer. iTrust is a web application written in Java, and the version of iTrust (v19 which was released in January 2015) that we used in the beginning of our Spring 2015 semester contained about 24.6 thousand lines of code. For that semester, 62 undergraduate students were enrolled in our software engineering class. These students worked in 15 teams and used the Eclipse development environment in their lab sessions. Four project milestones were defined: Lab1 was to index textual requirements, Lab2 was to index source code artifacts, Lab3 was to perform refactoring, and Lab4 was to handle changing requirements.

For the Spring 2016 semester, there were 103 students enrolled in our software engineering class. They worked in 25 teams and their subject system was a mobile application called Mapbox. We adopted v3.0.0 which was released in December 2015 to be the baseline for the students to work on. This version of Mapbox contained around 6,214 lines of code written primarily in Java. The students used Android Studio in their lab sessions for their development tasks on Mapbox. The four milestones for the Spring 2016 semester focused on delivering new features of Mapbox. They were: Lab1 was on implementing the feature of route navigation, Lab2 was on adding a new feature of making phone calls, Lab3 was refactoring, and Lab4 was hardening security (Cantrell, Dampier, Dandass, Niu, & Bogen, 2012) of Mapbox by dealing with changing requirements.

In the two semesters, a total of 160 lists of top five risks were collected from the 40 student teams. At each of the four milestone deliverables, each team was required to submit a lightweight reflection document, in which the top five risk items that they encountered in the current iteration had to be identified. We instructed the

students to rank their self-identified risks based on how seriously those items could negatively impact them move forward as a team. Note that relevant materials like risk exposure were taught and practiced early on in those semesters to provide the students with necessary background on risk management. It is also worth mentioning that the risks submitted as part of the last lab's reflection might be more perceived than those identified in other labs due to the end of the semester; however, the differences should not be significant, in our opinion, since the students were practicing active risk management in their agile software projects throughout the semester frequently in short cycles (3–4 weeks per milestone delivery).

## 4 Results and Analysis

In the results, we will be focusing on the risks identified by students involved in the agile teams enrolled in the two semesters of the software engineering course. While our analysis covers the collaborative and lean natures of these risks, we will first discuss how the risks collected in our study differ from those reported in prior work.

Within the literature, there are a number of different studies looking at top risks within software projects. In Table 1, we list the top 10 risks identified by notable researchers within the industry (Boehm, 1991; Reifer, 2002; Boehm, 2007). The lists shown in Table 1 focus on risks within the industry, however, when comparing these risks to risks that may be more likely within an academic context these risks are naturally going to be a bit different. For example, in Koolmanojwong and Boehm (2013), the authors argued the importance of requirements-related issues for teams working in an academic software engineering setting as not fully understanding the client's requirements would lead to project failures and expensive revisit of requirements later on in the development phase.

Based on our study's results, the students identified a number of risks specific to their context. Table 2 presents the aggregated risks alongside the top-10 risks identified in (Koolmanojwong & Boehm, 2013). In our analysis, an item was ranked higher based on the frequency of occurrence, that is, if the item appeared more times (irrespective of being ranked first, second, third, etc.) when we considered all the student teams' risk submissions.

When compared to Koolmanojwong and Boehm (2013), some top-ranked risks like requirements mismatch and personnel shortfalls identified by our students were similar. However, some risk items were more refined, e.g., the lack of overview was a manifestation of architecture complexity. In addition, we note when comparing our students' top risks with the other studies (Tables 1 and 2) that some risks were no longer ranked as highly by our students. Most noticeable of these was the requirement volatility. This risk only appeared a few times in the students' lists, indicating that new and changing requirements were possibly not perceived to be as such a high risk but more likely to be something to be expected in an agile environment.

**Table 1** Top-10 risk items with industrial relevance

Rank	Boehm (1991)	Reifer (2002)	Boehm (2007)
1	Personnel shortfalls	Personnel shortfalls	Architecture complexity, quality tradeoffs
2	Unrealistic schedules and budgets	Misalignment with business goals	Requirements volatility
3	Requirements mismatch	Unrealistic customer and schedule expectations	Acquisition and contracting process mismatches
4	User interface (UI) mismatch	Volatile technology (e.g., .NET, Persistence, J2EE, etc.)	Budget and schedule constraints
5	Gold plating	Unstable software releases (especially poor performance and frequent crashes)	Customer–developer–user team cohesion
6	Requirements volatility	Constant changes in software functionality	Requirements mismatch
7	Shortfalls in externally furnished components	Even newer methods and more unstable tools	Personnel shortfalls
8	Shortfalls in externally performed tasks	High turnover (especially of those personnel skilled in the new technology)	COTS (commercial off-the-shelf) complexity and shortfalls
9	Real-time performance shortfalls	Friction within the team (lack of leadership, overwork, etc.)	Technology maturity
10	Straining computer science capabilities	Unproductive office space	Migration complexity

#### ***4.1 Collaborative Nature of Risks***

We summarize our finding of the collaborative nature of students' perceived risks in this subsection; interested readers can refer to (Thota et al., 2017) for more detailed information. We classified the collected risk items into four categories based on risk identification and mitigation. The first category (C1) refers to those risk items that were identified individually and mitigated individually as well. An example of C1 is a student's insufficient background knowledge of a development environment (e.g., Eclipse or Android Studio). The second category (C2) contains the risks that were identified individually but mitigated collaboratively. A case in point is resolving merge conflicts in parallel development. The third category (C3) is composed of the risks that were identified collaboratively but mitigated individually. For instance, a number of requirements mismatches were reported by more than one team member but the resolutions were carried out predominantly solo. The fourth category (C4) has



**Table 2** Comparison of students' perceived top-10 risk items

Rank	Graduate-level courses (Koolmanojwong & Boehm, 2013)	Our junior-level software engineering courses	Example
1	Architecture complexity, quality tradeoffs	Team cohesion and communication	Team members do not respond to each other when support is needed
2	Personnel shortfalls	Software artifact dependencies	Certain requirements are requisite for implementing others
3	Budget and schedule constraints	Schedule constraints	Different course schedules limit team members' face-to-face meeting
4	COTS and other independently evolving systems	Task dependencies	Teams have difficulty in distributing workload and integrating individual contributions
5	Customer-developer-user team cohesion	Architecture complexity (especially lack of an architectural overview)	Team lacks an overall picture of the software as it relates to the lab deliverables
6	Requirements volatility	Personnel shortfalls	A single member needs to play multiple roles, sometimes with less competency
7	UI mismatch	Operational environment interoperability	Software debugged in one machine (e.g., Mac) has unexpected runtime behavior in another (e.g., Windows)
8	Process quality assurance	Requirements mismatch	Underspecifying requirements or making incorrect assumptions
9	Requirements mismatch	Process quality assurance	Team members cannot decide a sufficient unit testing level
10	Acquisition and contracting process mismatches	Unstable software releases	External components (e.g., libraries or APIs) become deprecated, causing unexpected crashes

Originally published in Chaitra Thota et al., "Students' Perceptions of Software Risks," 2017 ASEE Annual Conference, Columbus, Oh. © 2017 American Society for Engineering Education  
*Note* Although our student teams submitted their top-5 risk items at each milestone we have aggregated this to show the top-10 risk items for the purpose of comparison

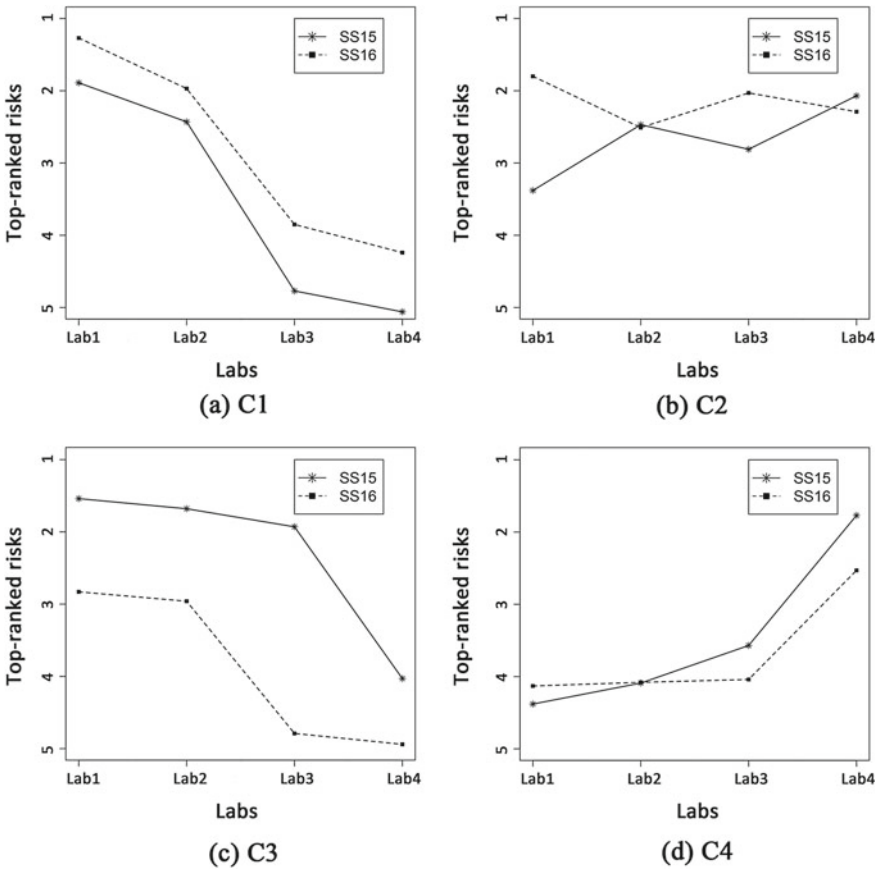
those risks that were both identified and mitigated collaboratively. Many instances of task dependencies (e.g., integration testing) fell into C4.

The temporal trends of each category's mean rank are plotted in Fig. 1. The mean score represents the average ranking of the risk items in a specific category where the students themselves provided the ranking (e.g., first, second, third, etc.) to imply that risk item's degree of negative impacts. There are quite a few C1 risk items related to the insufficient technical background. As shown in Fig. 1a, these risks were addressed (i.e., having lower ranks) in a rather quick manner as the students became familiar with the development environment, as well as the subject system (i.e., iTrust and Mapbox). A contrasting pattern was observed for the category C4. Figure 1d shows that it was not typical for C4 risks to emerge at the beginning of the software engineering course. However, once they were recognized, they tended to be ranked higher by the student teams, showing this type of risks' greater likelihood of occurrence and potential negative impacts. C2 risks, as shown in Fig. 1b, had a fluctuating temporal pattern, indicating the difficulty and uncertainty of jointly resolving the risks recognized only personally. Finally, for both semesters, the severity of C3 risk items depicted in Fig. 1c dropped as the semester went along. This implies that individually addressing the team-wide risks (C3) might be more effective than mitigating those risks in a joint manner (C4). As such, raising the awareness of the team-wide risks, without hinging on any particular mitigation or resolution, could be a valuable teaching strategy when teaching software development (Thota et al., 2017).

## 4.2 Lean Nature of Risks

Within the analysis of the risks, there is a strong focus on eliminating “muda” (waste). Muda can be classified into two types: non-value-adding but necessary for end-customers (Type I) and non-value-adding but unnecessary for end-customers (Type II) (Poppendieck & Poppendieck, 2003). Extending the muda types, a risk can be viewed as a waste if its actionability is low, i.e., non-value-adding but unnecessary for the students' agile team. In this chapter, we have adopted explicit semantic analysis (ESA) in order to automatically quantify the actionability of a risk. ESA (Gabrilovich & Markovitch, 2007) is a semantic relatedness method aimed at quantifying the degree to which two concepts semantically relate to each other, by exploiting different types of semantic relations connecting them. The main intent is to mimic the human mental model when computing the relatedness of words.

Our main motivation for adopting ESA is to *automatically* classify the level of actionability of the risks, and according to Gabrilovich and Markovitch (2007), Wikipedia provides reliable sources for implementing ESA. ESA represents the meaning of texts in a high-dimensional weighted vector of concepts derived from Wikipedia. In particular, given a text fragment  $T = \{t_i\}$ , and a space of Wikipedia articles  $C$ , initially, let  $[v_i]$  be the TFIDF weight (Baeza-Yates & Ribeiro-Neto, 1999) of the term  $t_i$ . Using a centroid-based classifier (Gabrilovich and Markovitch, 2007),



**Fig. 1** Ranking changes of risk categories. Originally published in Chaitra Thota et al., “Students’ Perceptions of Software Risks,” 2017 ASEE Annual Conference, Columbus, Oh. © 2017 American Society for Engineering Education

all Wikipedia articles in C are ranked according to their relevance to the text. Let  $k_j$  be the strength of association of term  $t_i$  with Wikipedia article  $c_j$ ,  $\{c_j \in c_1, c_2, \dots, c_N\}$  (where N is the total number of Wikipedia articles). Then the semantic interpretation vector  $S = [s_1, s_2, \dots, s_N]$  for text fragment T is a vector of length N, in which the weight of each concept  $c_j$  is defined as

$$s_i = \sum_{c_j \in c_1, c_2, \dots, c_N} v_i k_j$$

Entries of this vector reflect the relevance of the corresponding articles to text T. Finally, semantic relatedness between two text fragments is calculated as the cosine between their corresponding vectors. Among the different Wikipedia-based

**Table 3** Wikipedia categories included in and excluded from our ESA analysis

Included categories		Excluded categories	
Category	# Articles	Category	# Articles
Software	2,929,221	Human name disambiguation	2,634,660
Software design	2,645,816	County disambiguation	2,620,466
Software development	2,575,218	Days	2,495,113
Software quality	1,935,906	Geography by place	2,453,533
Human–computer interaction	1,756,037	Social media	2,391,391
Agile development	1,169,671	Business	2,133,730
Software requirements	1,125,125	Artificial objects	2,031,328
Computer programming	952,890	...	...

measures proposed in the literature, ESA has been proven to achieve the highest correlation with human judgment (Strube & Ponzetto, 2006; Mahmoud, Niu, & Xu, 2012; Mahmoud & Niu, 2015). In addition, ESA compares text fragments. This makes it a suitable approach for risk description analysis tasks. In this study, ESA is used to create a semantic relatedness vector  $w = \{w_1, w_2, \dots, w_{m-1}\}$ , where  $w_i$  is semantic relatedness between adjacent text fragments  $\{t_i\}$  and  $\{t_{i+1}\}$  in a risk  $r = \{t_1, t_2, \dots, t_m\}$ . Then, the actionability score of  $r$  is defined as:

$$a_r = \frac{\sum_{i=1,2,\dots,m-1} w_i}{m - 1}$$

The assumption here is that instead of using general terms which indicate several unrelated concepts, the actionable risk contains more terms which are related to the specific issues. Our operation, therefore, selects a certain group of Wikipedia categories and ignores many other categories. The inclusion criteria are based on software engineering, and the included categories, along with the number of articles in each category are listed in Table 3. In comparison, Table 3 also provides a sample of excluded categories. In addition to the software engineering categories, project-specific ones shown in Table 4 are also used. For the spring 2015 semester (SS15), categories related to software artifacts indexing and tracing (information retrieval, and natural language processing), as well as the development environment category (Eclipse plugins) are taken into account. For the spring 2016 semester (SS16), mobile app development categories (mobile apps and Android) and a couple of software security category (computer security and software testing) are included for the computation of a risk's actionability.

**Table 4** Project-specific categories included in our ESA analysis

SS15		SS16	
Category	# Articles	Category	# Articles
Information retrieval	1,994,217	Mobile apps	1,764,546
Natural language processing	1,631,682	Android	1,642,365
Eclipse plugins	673,241	Computer security	1,519,731
		Software testing	1,433,519

We illustrate our ESA-based risk actionability method with several examples:

- **Risk1 (incorrect assumptions; SS16):** *“When we met the first time for this Lab, we did not explicitly state how the program should function. Instead of going over the requirements carefully, we ended up making a lot of assumptions. This caused us to be crunched [for] time at the last moment (when we found out that our assumptions were partly incorrect). Next time [we] need to learn to communicate these requirements across before coding and starting the project. We need to read the Lab manual together to discuss it and also ask the TA for help if needed and establish the requirements. This was one of the weaknesses of the four of us as a team.”*
- **Risk2 (inadequate testing; SS15):** *“Moving forward, we are consciously aware of the need for test cases as well as improved quality in the software. Then we will be able to more confidently say that we are delivering a quality software product.”*
- **Risk3 (cross-platform compatibility; SS15):** *“Our customers will use different operating systems. Support all our customers is an important task.”*
- **Risk4 (new to Android development; SS16):** *“We need to understand how Android Studio Works. We also shall share our understanding and have group discussions.”*

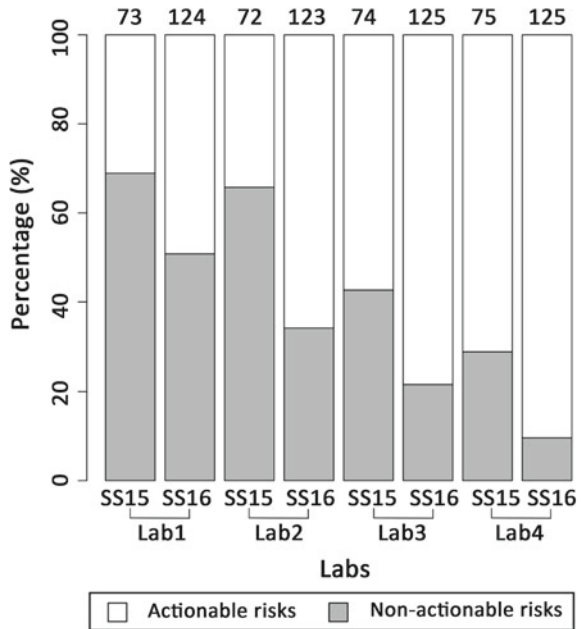
Using the software engineering and project-specific Wikipedia categories, the ESA-based actionability score of Risk1, Risk2, Risk3, and Risk4 is 0.77, 0.64, 0.21, and 0.42, respectively. Observing other ESA results suggests the threshold of 0.5 in terms of distinguishing actionable risks and non-actionable ones. Using this heuristic, Risk1 and Risk2 are regarded as actionable due to the sufficient descriptions of the risk itself and the specific plans for addressing the risk (e.g., establishing more correct requirements by working together with the TA, developing more and better test cases, etc.). In contrast, Risk3 and Risk4 exhibit significant vagueness (for example, which operating systems need to be compatible with which other ones, how to go about learning Android Studio, etc.). In order to use Risk3 and Risk4 as assets to move forward with the agile project, more work must be carried out. Compared to Risk1 and Risk2, these non-actionable risks would more likely lend themselves to be classed as muda, which lean intends to eliminate (Poppendieck & Poppendieck, 2003).

To test the impact of the excluded Wikipedia categories on the quantification of risks’ actionability, we randomly chose one such category and added its articles to

the ESA calculation: “Business” for SS15 and “Social media” for SS16. As a result, the actionability score of Risk2 and Risk4 changed to 0.47 and 0.53, respectively. The same ESA calculation was applied to all the students’ risks collected in our study. Using the same 0.5 threshold value would lead the classification of Risk2 and Risk4 to be actionable and non-actionable, respectively. This demonstrates the effectiveness of our selected categories and our adaptation of ESA in computing the risks’ actionability.

Figure 2, shows the temporal trends of the actionable and non-actionable risks. Because the number of risks differs from each lab, within the same semester as well as across the two semesters, we show in Fig. 2 the raw numbers on top of each bar. The comparisons are then performed by the normalized percentages. As the semester proceeds, it is evident from Fig. 2 that the proportion of actionable risks keeps increasing. Meanwhile, the trend of the decreasing proportion of non-actionable risks indicates the wastes were eliminated as the students’ agile teams worked through their milestone deliverables. We conclude that when risks are instrumented as an essential part of the agile deliverable, students tend to apply lean learning by eliminating non-actionable risks and identifying more actionable ones.

**Fig. 2** Distribution of actionable and non-actionable risks



### 4.3 Collaborative Meets Lean

Having shown the collaborative and lean natures of the risks in students’ agile teams, we now analyze the interactions between the two. We use Pearson’s chi-squared test for the interaction analysis. Chi-square fits our purposes because it is a statistical test applied to sets of categorical data to evaluate how likely it is that any observed difference between the sets arose by chance and the test is suitable for unpaired data samples (Gosall & Gosall, 2015). In our case, we are interested in the positive or negative association between risks’ collaborative nature (i.e., C1, C2, C3, and C4) and their actionality (i.e., actionable and non-actionable).

The results of the risks’ collaborative and lean analysis are shown in Table 5 where the positive/negative residual values indicate the positive/negative associations. However, the association is significant only when the residual value exceeds some threshold. According to Agresti (2007), a standardized residual having an absolute value that exceeds about 2 when there are few cells or about 3 when there are many cells indicates the significance of the association. Since our analysis involved few cells (for each semester, 8 cells were involved), we used the absolute value of 2 as the threshold. The bolded values in Table 5 then signal significance. For SS15, C3 (collaboratively identified, individually mitigated) risks have highly positive associations with being actionable, and meanwhile, C4 (collaboratively identified, collaboratively mitigated) risks correlate positively with being non-actionable in a significant way.

The results shown in Table 5 reveal the significant association between {C3, C4} and {actionable, non-actionable}, though the statistical significance holds only in one semester but not both. To consider both semesters, we further employed partitioning (Sharpe, 2015) to perform pairwise analysis of the association between {C3, C4} and {actionable, non-actionable}. The null hypothesis  $H_0$  is that there is no association of the collaborative and lean natures of the risks. The results, shown in Table 6, show that for SS15,  $\chi^2 = 10.486, df = 1, p = 0.001$ , and for SS16:

**Table 5** Chi-square analysis of the collaborative and lean risks

Category	Data	SS15		SS16	
		Non-actionable	Actionable	Non-actionable	Actionable
C1	Raw	47	43	102	43
	Residuals	+0.66	-0.66	-0.21	+0.21
C2	Raw	49	44	134	60
	Residuals	+0.79	-0.79	-0.77	+0.77
C3	Raw	<b>14</b>	<b>38</b>	29	20
	Residuals	<b>-3.56</b>	<b>+3.56</b>	-1.92	+1.92
C4	Raw	35	24	<b>88</b>	<b>21</b>
	Residuals	+1.72	-1.72	<b>+2.53</b>	<b>-2.53</b>

*Note* Items in **bold** indicate a significant association, + means a positive association, and - means a negative association

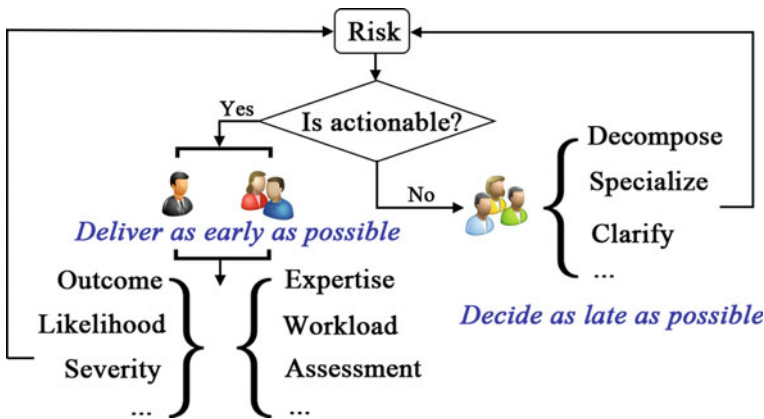
**Table 6** Partition-based chi-square tests where all the results are statistically significant

Category	Data	SS15		SS16	
		Non-actionable	Actionable	Non-actionable	Actionable
C3	Raw	14	38	29	20
	Residuals	-3.43	+3.43	-2.86	+2.86
C4	Raw	35	24	88	21
	Residuals	+3.43	-3.43	+2.86	-2.86

$\chi^2 = 7.086, df = 1, p = 0.008$ , where  $df$  stands for degree of freedom. These results lead to the rejection of  $H_0$ . We, therefore, conclude that risks impacting the students' agile team as a whole (C3 or C4) tend to be either highly actionable or non-actionable at all. We next discuss the implications of our results in the context of lean learning.

### 5 Discussion

When risks were instrumented as an essential part of the project's deliverable, the students' agile teams in our software engineering courses managed the risks in a collaborative and lean manner. Our findings could guide the ongoing effort of applying the lean principles in agile teaching, learning, and practice. We depict the implications of our work in Fig. 3. Risks represent potentials that lead to project failures, but risks themselves are neutral. Our results indicate that compared to the



**Fig. 3** Lean principles in agile risk management



risks identified by individuals, the ones recognized jointly by the agile team should receive more attention and therefore become more amenable to the lean principles.

While a risk has many properties, such as the likelihood of occurrence and severity, our work implies that the actionability should be considered as a prerequisite of applying the lean principles, especially the principle of elimination of waste. Our work further contributes an automated method based on semantic relatedness to quantify whether a risk's description is actionable or not. Although other methods such as risks' tree structures (Hoodat & Rashidi, 2009) can help differentiate the actionability, our method achieves the objective to a great extent.

In Fig. 3, if a risk is identified as actionable (for example, using our ESA-based method), then the lean principle, "deliver as early as possible", could be followed. The reason is that the risk item contains a sufficient level of details, which compared to non-actionable risks, could lead to more prompt actions. The actions, in turn, will increase the speed of the feedback loop. For example, in addressing the actionable Risk1 discussed in Sect. 4.2, gaining more correct, or at least more comprehensive, understandings about the functionality of the software by studying the requirements document carefully and by interacting with the customer representatives more frequently, this could be performed or delivered as early as possible. The challenge is to assign either an individual team member or a group of team members to deliver. From the perspective of risk management, certain aspects should be updated after the delivery, such as the outcome of risk mitigation, and in case the risk (for example, incorrect assumption about requirements) still exists, the updated likelihood and the severity of the risk should be updated. From the viewpoint of team management, factors such as expertise (who addressed the risk), workload (how much work was involved in mitigating the risk), and assessment (who validated and updated the risk) shall be maintained.

If a risk is recognized as less or not actionable, then Fig. 3 suggests that a different lean principle needs to be followed: "decide as late as possible." Moreover, the decision making shall be carried out in a collaborative way rather than individually. The progression can be made by decomposition if the risk covers too many concerns (e.g., quality tradeoffs among privacy, availability, and usability in mobile app development), by specialization if the risk is too general (e.g., the learnability of Risk4 discussed in Sect. 4.2), and by clarification if the risk's description is vague (e.g., which operating system or operating systems the customers require the software to be compatible with—Risk3 mentioned in Sect. 4.2). In addition to making the risk's description more detailed, options for mitigating the risk shall also be decided upon to improve actionability. As an example, Fig. 4 shows the risks related to UI mismatch between Mapbox as an implementation host and the "route navigation" as the to-be-implemented feature. In particular, there already exists a DoubleClickZoomHandler extending GestureDetector in Mapbox which zooms the map at a touch point by double-clicking. Should the students apply double clicking to set up the to-be-navigated point, the risk of UI mismatch would arise. Similar UI mismatch risk items include LongPressHandler, SingleTapHandler, and TapAndDragHandler in Mapbox. When "decide as late as possible" is practiced, the agile team shall be thorough about the ways such a risk could be addressed, e.g., by using a single

```

public class DoubleClickZoomHandler extends GestureDetector.SimpleOnGestureListener{

    private Animator mCurrentAnimator;

    private int mShortAnimationDuration;
    Point globalOffset;
    View _view;

    //Double click on the map view
    @Override
    public boolean onDoubleTap(MotionEvent e){
        float x = e.getX();
        float y = e.getY();

        try{
            zoomIn(_view,x,y);
        }catch (Exception ex){
            return false;
        }
        return true;
    }

    @TargetApi(Build.VERSION_CODES.HONEYCOMB)
    private void zoomIn(View view,float x, float y){...}
}

```

Fig. 4 Code snippet showing a UI mismatch risk in Mapbox

click to implement “route navigation” or by overwriting `DoubleClickZoomHandler` in Mapbox. While such decisions may always take long to be arrived at, we interpret the lean principle here to be “decide as thoroughly as possible”.

Our work has several limitations. First of all, the collaborative classification was done manually and in a post hoc manner. This, however, becomes less of a concern in Fig. 3 due to the focus on our implications on the actionability of the risks. While every risk can be automatically checked for its actionability, the collaborative analysis suggests that the ones impacting the team as a whole would be especially important for the agile and lean software development. Furthermore, the ESA-based method requires proper inclusion and exclusion of Wikipedia categories, which is currently performed manually. We believe that it is critical to the successful application of ESA to examine the actionability of the risks if one focuses on the risk categories that are specific to a software development project. Moreover, the chi-square correlation analysis that we performed in Sect. 4.3 treated the risks independent of the temporal properties. While this correlation analysis was informed by the temporal pattern of “eliminate waste” (cf. Fig. 2), having temporally sensitive analysis about the correlations may reveal further insights into how to apply the lean principles such as “deliver as early as possible” and “decide as late as possible”.

## 6 Conclusion

Risk is a fundamental vehicle for a software development project to progress and risk management is regarded as a critical skill in the software industry. We report in this chapter our two semesters' of teaching of a junior-level software engineering course, where students were grouped in agile teams to use risks to drive their milestone deliverables. Our analysis uncovered the collaborative and the lean natures of the risks in students' agile teams and further revealed the significant interactions between those natures. This allowed us to suggest specific lean principles for risk management in students' agile teams. Our future work includes improving the risk management teaching and practice in undergraduate software engineering courses, incorporating industrial-strength tools in teaching and learning risk management, and applying more lean principles in the context of agile risk management.

## References

- Agresti, A. (2007). *An introduction to categorical data analysis*. Wiley.
- Anslow, C., & Maurer, F. (2015). An experience report at teaching a group based agile software development project course. In *Proceedings of the ACM Technical Symposium on Computer Science Education, Kansas City, MO, USA* (pp. 500–505).
- Baeza-Yates, R., & Ribeiro-Neto, B. (1999). *Modern information retrieval*. Addison-Wesley.
- Beck, K., et al. (2001). Manifesto for agile software development. *Agile Alliance*. Retrieved from <http://agilemanifesto.org/>.
- Bhowmik, T., Niu, N., Wang, W., Cheng, J.-R. C., Li, L., & Cao, X. (2016). Optimal group size for software change tasks: A social information foraging perspective. *IEEE Transactions on Cybernetics*, 46(8), 1784–1795.
- Boehm, B. (1986). A spiral model of software development and enhancement. *ACM SIGSOFT Software Engineering Notes*, 11(4), 14–24.
- Boehm, B. (1991). Software risk management: Principles and practices. *IEEE Software*, 8(1), 32–41.
- Boehm, B. (2007). Top 10 software-intensive system risk items. In *Presentation at USC Annual Research Review*.
- Cantrell, G., Dampier, D., Dandass, Y., Niu, N., & Bogen, C. (2012). Research toward a partially-automated, and crime specific digital triage process model. *Computer and Information Science*, 5(2), 29–38.
- Carr, M. J., Konda, S. L., Monarch, I., Ulrich, F. C., & Walker, C. F. (1993). Taxonomy-based risk identification. *Technical Report*, CMU/SEI-93-TR-6.
- Cohn, M. (2013). A framework for evaluating agile risk management. Retrieved from <https://tcagley.wordpress.com/2013/10/01/a-framework-for-evaluating-agile-risk-management-daily-process-thoughts/>.
- Collofello, J. S., & Pinkerton, A. K. (1997). Integrating risk management into an undergraduate software engineering course. In *Proceedings of the 27th Annual Conference on Frontiers in Education, Pittsburgh, PA, USA* (pp. 856–860).
- Devedzic, V., & Milenkovic, S. (2011). Teaching agile software development: A case study. *IEEE Transactions on Education*, 54(2), 273–278.
- Emiliani, M. L. (2004). Improving business school courses by applying lean principles and practices. *Quality Assurance in Education*, 12(4), 175–187.

- Gabrilovich, M., & Markovitch, S. (2007). Computing semantic relatedness using Wikipedia-based explicit semantic analysis. In *Proceedings of the International Joint Conference on Artificial Intelligence, Hyderabad, India* (pp. 1606–1611).
- Gosall, N. K., & Gosall, G. S. (2015). *The doctor's guide to critical appraisal*. Chestire: Knutsford.
- Hanna, J. (2007). Bringing 'lean' principles to service industries. *Harvard Business School Working Paper*, No. 08-001.
- Holweg, M. (2007). The genealogy of lean production. *Journal of Operations Management*, 25(2), 420–437.
- Hoodat, H., & Rashidi, H. (2009). Classification and analysis of risks in software engineering. *International Journal of Computer, Electrical, Automation, Control and Information Engineering*, 3(8), 2044–2050.
- ISO. (2018). ISO (International Organization for Standardization) 31000—Risk management. Retrieved from <https://www.iso.org/iso-31000-risk-management.html>.
- Kamble, S., Jin, X., Niu, N., & Simon, M. (2017). A novel coupling pattern in computational science and engineering software. In *Proceedings of the International Workshop on Software Engineering for Science, Buenos Aires, Argentina* (pp. 9–12).
- Ker, J. I., Wang, Y., Hajli, M. N., Song, J., & Ker, C. W. (2014). Deploying lean in healthcare: Evaluating information technology effectiveness in US hospital pharmacies. *International Journal of Information Management*, 34(4), 556–560.
- Koolmanojwong, S., & Boehm, B. (2013). A look at software engineering risks in a team project course. In *Proceedings of the International Conference on Software Engineering Education and Training, San Francisco, CA, USA* (pp. 21–30).
- Mahmoud, A., Niu, N., & Xu, S. (2012). A semantic relatedness approach for traceability link recovery. In *Proceedings of the International Conference on Program Comprehension, Passau, Germany* (pp. 183–192).
- Mahmoud, A., & Niu, N. (2015). One the role of semantics in automated requirements tracing. *Requirements Engineering*, 20(3), 281–300.
- Niu, N., Bhowmik, T., Liu, H., & Niu, Z. (2014a). Traceability-enabled refactoring for managing just-in-time requirements. In *Proceedings of the International Requirements Engineering Conference, Karlskrona, Sweden* (pp. 133–142).
- Niu, N., Brinkkemper, S., Franch, X., Partanen, J., & Savolainen, S. (2018). Requirements engineering and continuous deployment. *IEEE Software*, 35(2), 86–90.
- Niu, N., & Easterbrook, S. (2007). Analysis of early aspects in requirements goal models: A concept-driven approach. *Transactions on Aspect-Oriented Software Development, III*, 40–72.
- Niu, N., Savolainen, J., Niu, Z., Jin, M., & Cheng, J.-R. C. (2014b). A systems approach to product requirements reuse. *IEEE Systems Journal*, 8(3), 826–827.
- Poppendieck, M., & Poppendieck, T. (2003). *Lean software development: An agile toolkit*. Addison-Wesley.
- Radnor, Z., Walley, P., Stephens, A., & Bucci, G. (2006). Evaluation of the lean approach to business management and its use in the public section. *Scottish Executive Social Research*.
- Reifer, D. (2002). Ten deadly risks in internet and intranet software development. *IEEE Software*, 6(2), 12–14.
- Rico, D. F., & Sayani, H. H. (2009). Use of agile methods in software engineering education. In *Proceedings of the Agile Conference, Chicago, IL, USA* (pp. 174–179).
- Ropponen, J., & Lyytinen, K. (2000). Components of software development risks: How to address them? A project manager survey. *IEEE Transactions on Software Engineering*, 26(2), 98–112.
- Schmidt, C., Dart, P., Johnston, L., Sterling, L., & Thorne, P. (1999). Disincentives for communicating risk: A risk paradox. *Information and Software Technology*, 41(7), 403–411.
- Schroeder, A., Klarl, A., Mayer, P., & Kroiss, C. (2012). Teaching agile software development through lab courses. In *Proceedings of the IEEE Global Engineering Education Conference, Marrakech, Morocco* (pp. 1–10).
- Sharpe, D. (2015). Your chi-square test is statistically significant: Now what? *Practical Assessment, Research & Evaluation*, 20(8), 1–10.

- Strube, M., & Ponzetto, S. (2006). Wikirelate! Computing semantic relatedness using Wikipedia. In *Proceedings of the National Conference on Artificial Intelligence, Boston, MA, USA* (pp. 1419–1424).
- Thota, C., Niu, N., Wang, W., & Purdy, C. C. (2017). Students' perceptions of software risks. In *Proceedings of the ASEE Annual Conference, Columbus, OH, USA*, Article No. 18053.

**Part V**  
**Using Agile and Lean Methods to Teach**  
**Software Development**

# Applying Lean Learning to Software Engineering Education



Robert Chatley

**Abstract** In this chapter, we describe the ways that we have applied lean and agile techniques to teaching software engineering at Imperial College London. We give details of the structure and evolution of our programme, which is centred on the tools, techniques and issues that feature in the everyday life of a professional software developer working in a modern team. We also show how aligning our teaching methods with the principles of lean software delivery has enabled us to provide sustained high-quality learning experiences. We examine two different types of course in detail: first, a ‘traditional’ lecture course, where we transformed the way that course is taught and assessed, aiming to create tighter feedback loops, and second a project-based course where we ask students to put agile methods into practice themselves, working in teams to build a substantial software system over a number of months. We describe concretely how we run and structure these courses to set up effective learning experiences.

**Keywords** Software engineering · University · Automation · Feedback  
Project-based learning · Peer-instruction

## 1 Introduction

Lean and agile methods are prevalent in industrial software engineering today (Papathoecharous & Andreou, 2014). Scrum, Kanban and eXtreme Programming (XP) are all common in software development organisations, helping teams to develop software iteratively, in reliable and predictable ways, whilst responding to changing requirements in a fast moving world. In university Computer Science departments, we are training the next generation of software engineers, and it is therefore important that we teach these methods to prepare students for their future working lives.

---

R. Chatley (✉)

Department of Computing, Imperial College London, London, UK  
e-mail: [rbc@imperial.ac.uk](mailto:rbc@imperial.ac.uk)

© Springer Nature Singapore Pte Ltd. 2019

D. Parsons and K. MacCallum (eds.), *Agile and Lean Concepts for Teaching and Learning*, [https://doi.org/10.1007/978-981-13-2751-3\\_14](https://doi.org/10.1007/978-981-13-2751-3_14)

285

Many universities and other higher education institutions are striving to bring modern industrial software development techniques into the classroom, and like any such institution, Imperial College London has been faced with the challenge of updating and evolving its software engineering education to prepare its students for modern industrial careers. Keeping pace with rapid changes in industrial practice has required changes in the way software engineering is taught. This includes teaching modern development methods and giving students hands-on experience of putting those methods into action through practical work (Anslow & Maurer, 2015; Kropp & Meier, 2014). This evolution has not been easy but, through continuous experimentation and iterative improvement, we believe that we have evolved a software engineering programme that strikes a good balance between teaching, learning, and assessment.

Given that we are teaching lean and agile methods, and believe that they have positive effects on software engineering practice, it seems natural that we use them to inform our teaching practice too. If we are looking to reduce waste, and to improve quality and feedback in our educational systems, can we apply the principles and practices that we teach to the teaching itself?

Although our own courses are focussed on software engineering, we believe that many of the lessons we have learned are transferable to other disciplines.

## ***1.1 Perspectives on Teaching***

In order to discuss the approaches that we have tried, we will borrow some vocabulary from Mark Guzdial's 2015 book 'Learner-Centered Design of Computing Education' (Guzdial, 2015). Guzdial gives us three useful terms to describe different types of learning experience. The first is *transmission*, which describes the classic lecture situation. An expert holds a body of knowledge and tries to transmit it to a—hopefully—attentive audience. This is typically a one-way interaction between one teacher and many learners.

The second perspective is *apprenticeship*, which we use to describe a learning experience focussed on the development of skills rather than theoretical knowledge, most likely through kinaesthetic learning and practical exercises. You can imagine this in a setting like a cookery class, where each student can practice a recipe repeatedly until they have mastered a dish.

The third perspective is *developmental*, which describes a personalised learning experience without a set curriculum. It focuses on taking the learner from where they are to somewhere more advanced, in a particular direction depending on their strengths and weaknesses. This sort of individual tuition works well in a situation like a piano lesson, but it is hard to replicate it with a lecture class of 150 students.

Unfortunately, we do not have the resources in our university to offer individual tuition and personally tailored programmes for every student taking Computer Science, perhaps as a student at a music conservatoire might experience. However, we



will discuss how we have tried to blend these three approaches in order to improve on a style of teaching purely based on weeks of transmission followed by final exams.

## 1.2 *The Rest of This Chapter*

In the remainder of this chapter, we will illustrate how we have transformed two different types of courses to increase the value of the learning experience, and to incorporate more frequent, high-quality feedback. We move from courses primarily based on *transmission* to courses that focus on the development of skills through an *apprenticeship* model, and also incorporate individual and small group tuition from instructors and peers, moving towards more *developmental* education. In Sect. 2, we look at the evolution of a traditional lecture course, and in Sect. 3 we describe how we support different types of project-based learning. Section 4 discusses possible challenges for future adoption of similar techniques in more courses and at larger scales. We also give some qualitative feedback taken from our student survey, which is conducted across all students, anonymously, at the end of each term of study.

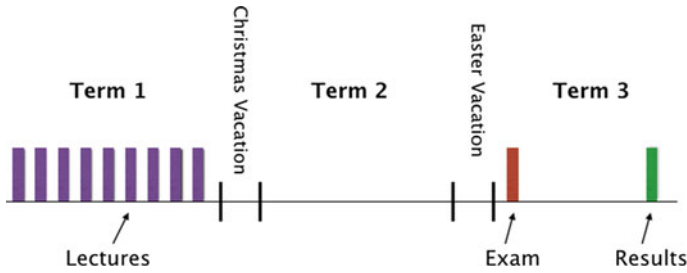
## 2 Lecture Courses

Within Imperial's Computing curriculum, we have a second-year undergraduate module called *Software Engineering Design*. The content of this module concerns methods, tools, and techniques for the development and deployment of large-scale software systems that are robust, well-engineered and easy to maintain by design. In an earlier incarnation of the course, the material concentrated on notation, formal specification languages and catalogues of *design patterns* (Gamma, Helm, Johnson, & Vlissides, 1995). This meant that students would learn a range of ways to document and communicate software designs, but these were not tied to a particular implementation language. Much of the material was thus taught 'in the abstract' and the students did not get much opportunity to put their theoretical knowledge into practice. The following comment in our student survey typified concerns that this was not the best approach:

Would have preferred design patterns to be practiced more in lab exercises, ... the patterns I understood best were the ones for which I wrote and tested actual code...

We wanted to find a way to move the focus from learning theoretical knowledge to applying and demonstrating practical skills. We hoped that this would not only improve the students' experience of the class, but also provide them with a more valuable learning experience.

Historically, teaching in this class was based largely on *transmission*. Students attended lectures twice per week throughout the autumn, took other modules during



**Fig. 1** The structure of a traditional lecture course at Imperial, with lectures over the autumn term, and examinations in the summer. Figure © 2017 IEEE. Reprinted, with permission, from proceedings of 2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)

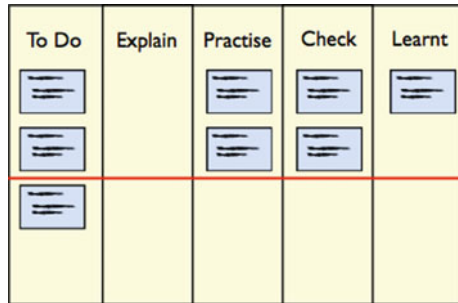
the spring, and then had their examinations after Easter (see Fig. 1). There were tutorial classes alongside the lectures, usually with paper-based exercises, but typically only the most diligent students kept up with the exercises week by week and most left them to use as revision aids come exam time.

This approach is completely at odds with the typical delivery cycle of a modern industrial software project. The feedback cycle is very long, and a large amount of work is in process before we get to the ‘quality assurance’ stage. Only when we get the exam results do we really know whether we have taught the students effectively. We can think of the course as starting out with a long list of requirements for things that students should learn—a syllabus—and that we then go into a phase of transmission, after which we check the results. There is no iteration or incremental delivery—it is one big batch.

In modern software development projects, we typically strive to reduce batch size, with the aim of decreasing cycle time, decreasing risk, and increasing quality. One mechanism by which we might do this would be to employ Kanban, a lean method that focusses on flow through a system, and by using it we can aim to maximise throughput and minimise cycle time (Anderson, 2010). In software development, we want to minimise the time between someone having an idea for a feature and prioritising it, and that feature being working software in the hands of the users. In learning and teaching, we instead want to minimise the time from introducing an idea, to having a student internalise it, to verifying that their understanding is correct.

One of the tools of a Kanban practitioner is to visualise the workflow. In a software project, this is typically done with a physical or virtual ‘card wall’, divided into columns for the different phases that each piece of work needs to go through. The different columns map the *value stream* (Rother & Shook, 2003). Typically the board is divided into columns representing the ‘backlog’ of upcoming tasks, those that are in analysis, those in development, those being tested, and those ready for release, or released. Cards representing separate tasks are moved from column to column as work on them progresses. Similar boards are also often used in other agile methods such as Scrum and XP, but where in those methods the board is an information radiator to help to display the current state of the team’s work, within a regular

**Fig. 2** The value stream of learning, which maps the different phases of learning. Figure © 2017 IEEE. Reprinted, with permission, from proceedings of 2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)



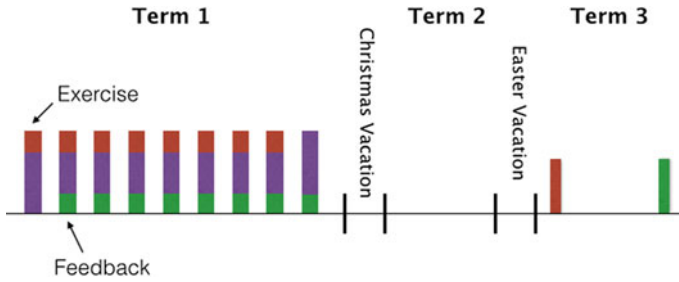
delivery cadence (e.g. a 2-week iteration), in Kanban the board is used as a tool to define and optimise the flow of work through the system. The key idea is to use the current state of the work to decide what to do next, and always to ‘pull from the right’, so that we concentrate on getting individual pieces of work finished before starting new ones (Ottinger, 2015). This way we focus on completion and keep the work in process low. A limit can be placed on the number of pieces of work that may appear in any column at once in order to enforce this focus on finishing.

We take this idea and redraw the columns on the board to form a value stream of learning. Here we list the items on the syllabus as our backlog—‘to do’—and then have columns for ‘explain’ (transmission), ‘practice’ (apprenticeship), ‘check’ and ‘learnt’ (see Fig. 2). If we follow the ‘pull from the right’ mantra, then we want to get each item over to the right-hand side as quickly as possible. That means that we aim to do a minimal amount of transmission on each topic before the students get to practice in a hands-on exercise, and then verify the quality of their learning, obtaining feedback before we move further on in the syllabus.

Putting this into practice, we first tried the common approach of adding a small project as coursework part way through the term. However, as it took a couple of weeks to complete the project, and about the same again to get all the assignments marked up and graded, it was pretty much the end of the course before the students got their feedback. There was a wide variation in how students chose to approach the design project we gave them. Those who were more dedicated and had understood well tried out a lot of different ideas and added many features. Those who had not understood well did much less or did the wrong thing. If anything, rather than making sure that everyone had learnt the material, it seemed that we had widened the gap between the stronger students and the weaker ones. We needed something better.

### 2.1 Reducing Cycle Time

In order to give more guidance, and earlier feedback, we changed from asking students to design a whole system to asking them to consider individual design choices in different situations, and examining how implementing something one way or another



**Fig. 3** Revised course structure, with a weekly cycle of assessment and feedback. Figure © 2017 IEEE. Reprinted, with permission, from proceedings of 2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)

would affect the future maintenance of the system. In terms of the assignments that were set, we moved from ‘design a system with the following requirements, discuss the design choices you made’, to a set of weekly smaller coding exercises of the form ‘Add feature X to this system by using design pattern Y. Now try design pattern Z. What are the trade-offs?’. By carefully constructing a number of small scenarios to work through one-by-one, we ensured that each student had the same design issues to think about, and by making them into coding examples students got a much more hands-on, kinaesthetic learning experience.

Fitting with the weekly nature of the university timetable, this led to a weekly cycle of assessment (see Fig. 3). A new topic is addressed each week with an associated assignment, and students submit their solution later the same week. Grades and feedback are then returned within three or four working days, i.e. before they submit their next assignment.

The obvious problem with weekly assignments is the volume of grading and feedback required. Because of the limited teaching resources that institutions generally have to work with, the temptation is to reduce the frequency of assignments, e.g. to once every 2 weeks, in order to be able to deliver feedback ‘at scale’. However, this is at odds with what we are trying to achieve. Relating this again to the conditions that apply in a software development project, often we strive to release software more frequently, but integrating and testing new code requires a lot of time and effort. XP promotes adopting a process of *continuous integration* (Duvall, Matyas & Glover, 2011), through which we tend to find that doing these things more often causes us to streamline processes, remove waste, and often apply automation. As Martin Fowler often says ‘if it hurts, do it more often’ (Fowler, 2011).

## 2.2 Peer Coaching

One change that made a big difference to both student learning and marking load was to encourage students to pair-program, which has been shown to be highly effective in

a classroom environment (Williams & Upchurch, 2001). We have found that students enjoy the experience of working with a colleague—a class survey showed that from 148 students, 119 declared that they found learning to through pair-programming to be a good experience, 18 were neutral, and just 9 stated that they preferred to work individually. Students get to practice pair-programming, which is an industry-relevant skill, but not something that necessarily comes naturally to everyone; becoming good at it is difficult and requires work. The students get to coach each other and help each other to learn and understand. By engaging them in pair-programming, we had effectively set up a network of peer coaches—a *developmental* learning style personalised to each individual. Although we are aware of studies that show that constructing pairs by matching weak and strong students perhaps produces more learning, in this case, we allowed them to work with whomever they liked as we wanted to smooth the path to adoption. We may experiment with pre-selected pairs in future. Last, a major benefit in terms of giving weekly feedback on assignments was that pair-programming reduced the number of submissions from 150 to 75!

### 2.3 Automation

A key to reducing the burden of assessment and feedback has been to add automation where possible. This ties back to the engineering practices of XP where automation is used to allow testing of software to be done quickly, repeatably and reliably. Our approach here has been to provide tools that enable students to test their exercise solutions as they work, to detect basic problems early and allow students to fix them before they submit their work. From the first week of their first year, students learn to use version control through Git and GitLab.<sup>1</sup> When they start a coding exercise they clone a repository to obtain skeleton files that form a starting point and are encouraged to work in small steps, committing each change as they go. When they submit their work for assessment, what they actually submit is a Git commit hash corresponding to the version to be marked. We have also implemented a software testing tool the ‘Lab Test System’ (LabTS), which allows students to view and test each version of their code themselves (see Fig. 4).

For first-year courses, we provide a (partial) test suite that students can run against their code, to check the correctness of their solutions. However, later, when learning about software design, we do not want students to follow the same approach. Providing a set test suite has a consequence of defining an API that the students need to implement. In our design exercises, we want them to design their own API as part of the exercise, and to write their own automated tests against that API. Writing automated tests and utilising test-driven development is a key skill that we want to instil at this stage of the students’ education.

As a mechanism to encourage students to write their own tests, we use LabTS to check a test-coverage metric, with a coverage threshold that we deem appropriate

---

<sup>1</sup><https://about.gitlab.com/>.

The screenshot shows the LabTS interface for a repository named 'Grumberg, Daniel's repository for Camera'. At the top, there are options to 'Switch User' (currently 'Abdalla, Taha (a2813)') and 'Chatley, Robert (admin)'. A 'Milestone' dropdown is set to 'Tue 2'. The main content is a table with the following columns: Commit Hash, Commit Message, Commit Date, Status, Result, Request Test, and Cate. The table contains 11 rows of commit data. The first row, commit 'e0ee2527', shows a 'Test Complete' status with a score of 5/5 and a green 'Submitted to CATs' button. The remaining 10 rows show 'Currently Untested' status with a score of 4/5 and red 'Submit Late to CATs' buttons. At the bottom of the table, there is a blue button that says 'View this repository on GitHub' and a link to 'Lab exercise'.

Commit Hash	Commit Message	Commit Date	Status	Result	Request Test	Cate
e0ee2527	Fixed the checklist issues. DG	2015-10-25 15:03:54 UTC	Test Complete	5/5	Request test	Submitted to CATs
3ca1503e	Tested and implemented sensor doesn't go off when camera turns off and data is s...	2015-10-23 22:08:18 UTC	Test Complete	4/5	Request test	Submit Late to CATs
0f41073f	Tested and implemented sensor doesn't go off when camera turns off and data is s...	2015-10-23 21:49:15 UTC	Currently Untested		Request test	Submit Late to CATs
3492c23a	Tested and implemented that data from sensor gets written to memory card when th...	2015-10-23 20:10:10 UTC	Currently Untested		Request test	Submit Late to CATs
997d38bb	Testes and implemented that nothing happens when the shutter is pressed when the...	2015-10-23 19:56:24 UTC	Currently Untested		Request test	Submit Late to CATs
c4cc8599	Tested and implemented that the sensor powers down when the camera's power is turned off. DG AK	2015-10-23 19:04:49 UTC	Currently Untested		Request test	Submit Late to CATs
de3560b	Added test to check if the camera is off on creation. DG AK	2015-10-23 19:00:29 UTC	Currently Untested		Request test	Submit Late to CATs
99e5b73b	Implemented the testy and feature sensor gets powered up when the camera turns on. DG AK	2015-10-23 18:45:03 UTC	Currently Untested		Request test	Submit Late to CATs
a6503829	Initial files for exercise	2015-09-05 12:06:31 UTC	Currently Untested		Request test	Submit Late to CATs

Fig. 4 Screenshot of our LabTS system—a web-based tool that allows students to run automated tests on each iteration of their exercise solutions

for that week’s exercise. LabTS gives each submission a score out of 3: 1 point if the code compiles, 1 point if all the tests the students have written pass, and 1 point if these tests meet the code coverage threshold and the code passes some basic layout and formatting checks. The exercises for our second-year design course are in Java, so we use a Gradle<sup>2</sup> build to choreograph the compilation, testing and other checks. We configure Gradle plugins to check code formatting against a given style guide, and to measure test-coverage. LabTS is then set up to run this Gradle build against each submission and report the results. Usually, if a LabTS test run does not score 3/3, it is relatively easy for the student to see what they need to do to make up the remainder of the marks. We put a policy in place for the class that if a solution does not score 3/3 on LabTS then a human marker does not need to look at it.

With this system in place, we cannot yet dispense with the human markers, but they can give more nuanced feedback on issues of design, and should not have to pick up on basic points about compilation, style, or test-coverage. Even the simple application of checks on code layout and style mean that by the time a person looks at the code, it is laid out in a way that is easy to read. This makes the most of the marker’s time by allowing them to focus on more subtle design issues, and not to waste time commenting on things that can be detected automatically. For our cohort of 150 students, working in pairs, with a team of 5 or 6 markers, we can mark and give feedback with about 2 hours of effort per marker per week, which allows us to sustain weekly feedback throughout the course.

<sup>2</sup><https://gradle.org>.

## 2.4 Summary

Changing the delivery format of this course from knowledge acquisition through transmission in lectures, to a focus on skills development through apprenticeship and practical exercises, together with the developmental support of peer coaching through pair-programming seems to have been a great success. To enable consistent progress and feedback through weekly exercises we had to solve the problem of scaling our feedback mechanisms, and have used automation techniques, as well as paring back the exercises to really focus on the core message, to make this manageable. The concrete nature of the exercises results in students feeling that their coding skills as well as their design skills are improved by completing them. They also appreciate getting weekly feedback on their work. The following comments from recent student surveys are quite typical:

A well-structured and engaging course, which I could immediately benefit from as it helped improve the quality of my code and Java knowledge.

I liked that I had to submit the tutorials every week, otherwise I would not have done them.

The weekly cycle of assessment and feedback now works really well. The small batch size and short turnaround time means that students are motivated to do the weekly assignments and this gets them to practice and to improve. Although we have not been able to automate marking completely—this seems like a grand challenge—we have found that a team of five people can now complete the feedback for the entire class in around 2 hours each week.

## 3 Project-Based Courses

Another key feature of Imperial's Computing curriculum is team-based practical projects. Team working is an essential component of any software engineering programme and is a key skill that many employers look for when hiring graduates. Modern software development methods focus on teamwork and collaboration for the development of software, and we feel that the best way for students to learn these is to experience them practically in project-based courses. Again, we aim to focus on *apprenticeship* and *developmental* learning, allowing students to learn for themselves and from each other through solving practical problems.

In our programme, students get experience of working in small groups from as early as the first year, but we increase the structure and process around the management of project work, along with the scale of the projects they tackle, as they progress through their education. First-year projects are left fairly freeform, with small groups and fairly short timelines, for the students to manage, however, they see fit. After that, we begin to introduce more structure as their projects grow, to allow them to experience something closer to an industrial development environment. Rather than concentrating on transmitting them the relevant theory, we focus on creating a learn-

ing experience where the structure in which projects are assessed naturally promotes an agile way of working.

### ***3.1 Second Year—Web Application Development Projects***

By the end of their second year of study, computing students should have learned the skills and knowledge needed to build a complete application. At the end of the summer term (the end of the academic year), we give them the opportunity to exercise these skills in a group project, and through this introduce some elements of agile development methods (such as Scrum or XP). The aims of these second-year group projects are to...

- explore user focussed design and development
- experience and practice an agile method in a small project
- apply software development tools and techniques
- develop team work

These projects are done in groups of four students, and run full-time over a period of 4 weeks, which we structure as four 1-week iterations. Each project team develops a web or mobile application of their own design to solve a problem that they have identified. They make the product decisions; they do not have an external customer. To emphasise an iterative approach we run the projects with the following structure:

- 1-week iterations, with groups required to demo their software every Friday.
- Demos are assessed in a lightweight way focussing on: product increment (have they delivered anything this week?) and user research (have they gathered feedback from their target users?).

Rather than marking the project entirely at the end, a proportion of the marks is available each week, so that sustained, iterative progress is rewarded—a big bang release at the end is not. We also want to steer the students towards quantitative evaluation of their own work through meaningful experiments with users. The Friday demos allow groups to demonstrate both their newly developed features and the results of that week's user trials in a short informal meeting. Each group gets 5–10 min to meet with tutors in the computer lab and showcase their latest work. We do not require a big final project report, just a few lightweight deliverables to document the purpose of the application, the overall technical architecture, and the use of appropriate development techniques.

The emphasis is on creativity, user experience, rapid iteration and vertical slicing of development. We do not go into the details of any particular project management practices or enforce that students must follow them. The weekly demos mean that the teams must integrate their features to have a working system each week, so they naturally follow a process of continuous integration and frequent release, without us requiring those practices explicitly. Our experience shows that if we require a team to demonstrate, for example, specific Scrum practices, then they tend to show them lip



service, and write a report telling us what they think we want to hear, without really feeling the benefit of the practices in their projects (especially when projects are relatively small scale, as university projects tend to be). Hence, we focus on regular delivery of working software above all else.

These projects are a fun way to finish the year, allowing students to apply their knowledge and build a product. They also serve as a warm-up for the larger projects that they will undertake when they return in the third year.

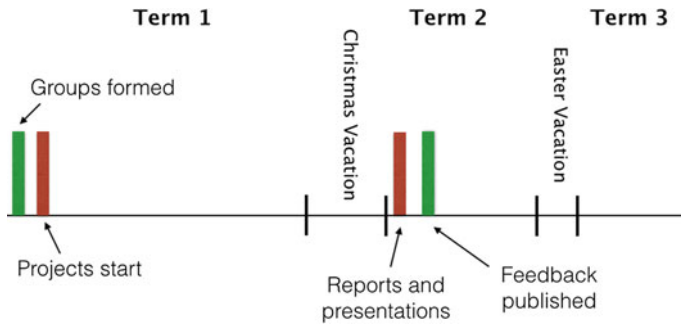
### ***3.2 Third Year—Software Engineering Group Projects***

By their third year of study, computing students should be in the position where they can use their skills to engineer a substantial software system. We want to give them the opportunity to exercise and develop these skills in a relatively large group project over the course of a few months. Where the second-year projects have a small team, a short timescale, and creative freedom to develop whatever they want, third-year projects have bigger teams, a longer timescale, and a customer relationship to manage. All of this naturally requires more conscious management, and so we can support this by encouraging teams to follow agile methods more explicitly. The aims of these projects are to...

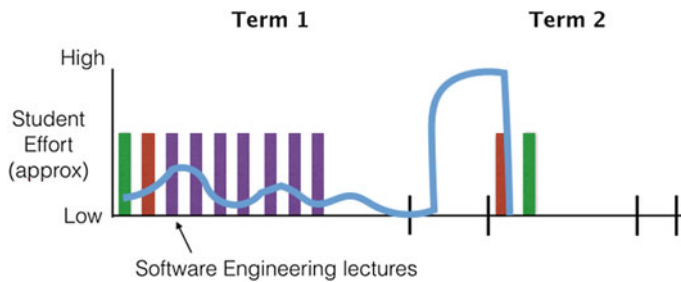
- apply software engineering tools and techniques
- apply management techniques for software projects
- develop a complex system for and with a customer, with a particular user in mind
- improve team work

Projects are done in groups of 5–6, between October and January (see Fig. 5), to a brief suggested by an academic supervisor (or in some cases an external company) acting as a customer to set requirements and guide the product direction. Students do not work on the project full-time, but alongside their lecture courses, including a Software Engineering course designed to support the project work. Each group has a different brief, but all are aiming to build a piece of software that solves a particular problem or provides a certain service for their users. Recent examples include an open-source implementation of Microsoft’s RoomAlive (Jones et al., 2014), systems for estimating heart rate based on video or speech recordings, and verifying product provenance using BlockChain technology.

The aims from an educational point of view are to build the students’ skills in teamwork and collaboration and to put into practice software engineering techniques that support this kind of development work.



**Fig. 5** Schedule for third-year Group Projects—project duration is 3 months. Figure © 2017 IEEE. Reprinted, with permission, from proceedings of 2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)



**Fig. 6** Perceived effort curve for students during Group Projects (in blue). Figure © 2017 IEEE. Reprinted, with permission, from proceedings of 2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)

### 3.3 Sustainable Pace

We run these projects during the first term of the academic year when the students have just returned from their summer vacation. Something we observed in previous years was that students tended to leave the bulk of the work on their project to later in the term, with a big effort spike as the deadline neared—not working at the sustainable pace that we would hope to see in an agile project (see Fig. 6).

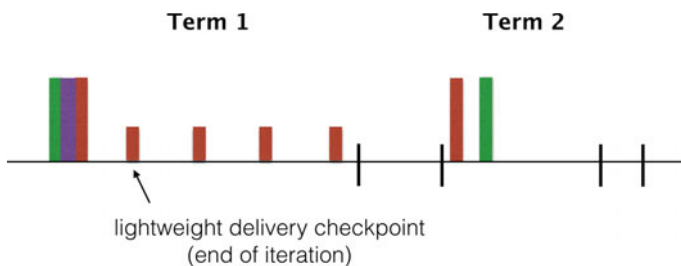
To encourage a more sustained pace of work, we introduced structured, time-boxed iterations. In the second-year projects described previously, students completed four 1-week iterations, and now for third-year projects we expect them to run their project as four 2-week iterations, through the autumn term, with a checkpoint at the end of each iteration where they demo their progress to their supervisors. More advanced teams could complete eight 1-week iterations if they prefer. In the 9th week, they should take a break for exams, and this then gives them the Christmas vacation to polish any final features, write up their reports and prepare their

presentations which are given in January. The aims of the following structure and deliverables are...

- to encourage students to do more work on the projects earlier in the term
- to encourage sustained, iterative progress on projects
- to encourage projects to ‘build a system that lets person X achieve Y’, rather than research projects
- to remove any deliverables (such as reports, etc.) that do not directly add to the project

We wanted to find ways to get the students to start earlier. To this end, we have now phased out the lectures, and instead give them introductory talks introducing the structure and goals of the project on the first Monday of the new term, have them form groups on Tuesday, select projects on Wednesday and complete allocations of projects to groups by the end of Thursday. Given the way that our timetable works, this then gives them a couple of clear days to make plans and get started on the project before their lecture courses start the following week. We then run the four 2-week iterations from weeks 2 to 9 of the term (Fig. 7).

In forming project groups, so far we have let students select their own teammates. Although, as with pair-programming, there are suggestions that learning can be improved by carefully selecting students of varying academic strength and mixing them together in each group, we felt that overall, the experience of restructuring and improving our projects would be smoother if we did not have students complaining that they were not able to work with their friends. This is something that we might revisit in future, but for the moment we plan to leave the groups as self-selecting—we only constrain the team size. Before our focus on agile methods, we specified that each team should appoint a team leader, but mandating this did not seem to fit well with the collaborative nature of agile methods, so we have left it to teams to do as they think best. If they choose to do Scrum, they should appoint a Scrum Master, but that is someone who is responsible for the execution of the Scrum process and as such very different from a team leader who makes the decisions and allocates work.



**Fig. 7** Lightweight end-of-iteration checkpoints every 2 weeks. Figure © 2017 IEEE. Reprinted, with permission, from proceedings of 2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)

### 3.4 *Customer Relationships*

The aim of these projects is to deliver a particular piece of software, rather than to conduct an investigation in a research area. Particularly in the case of some academic supervisors, we have had to steer them quite strongly to propose suitable projects. We found that projects that were very investigative, or required a lot of up-front reading of research papers, were generally not very compatible with our aims of delivery working software early and often.

To help to gather appropriate project briefs, we provide a template for the potential supervisors and customers to complete. The aim of using the template is to provide some consistency across project proposals, to make sure that the students have all the information they need in making their selection, and to make sure that project proposals fit well with the learning objectives in agile software development. Our project proposal template is as follows:

Please make sure that you have a particular piece of software in mind for the students to build. Your proposal should specify the following:

**The project title is...**

**General background...** a brief description of the context and purpose—a couple of paragraphs

**The target user is...** identify a particular user (or class of users)—be specific

**The system should allow the user to achieve...** be specific about the capability that the application should provide—what can you do when it exists that I can't do now.

**Any technical (or other) constraints...** e.g. this must be an iPhone app; this must run on a video wall; you must integrate this C++ library; you must collaborate with this external company, etc.

### 3.5 *Lecturing Versus Coaching*

Having completed their second-year projects, students should already have experience of working in an iterative way for a small project. Now, having reduced the *transmission* of Software Engineering materials through lectures, we just provide a recap lecture during the first week of term, reminding them of the agile techniques they used in their previous projects. Additional material giving more detail about specific methods (Scrum, XP and Kanban) is provided online. We also provide online material on continuous integration and delivery techniques, as well as case studies of commercial implementations of agile methods at companies like Spotify who are quite open (Kniberg, 2014) about the methods that they use.

Rather than giving further lectures, we now provide individual consulting time to all groups by holding office hours. Groups can book slots to get specific advice on any problems they are having, particularly around the areas of project management, evaluation, or other software engineering matters relevant to their project. Making

these office hours optional meant that only the keen groups (often those not requiring much help) took up the opportunity. In light of that, we now mandate that each group must arrange at least two consultations over the course of the project, one during the first four weeks, and one during the second four weeks. The consultations are 30 min each, and we have around 30 groups, so this is 30 hours worth of work for the tutor over an 8 week period. We ensure that the tutor running these consultations is an experienced industry professional with experience of applying various agile methods in many different software engineering contexts. This way we can give high value, context-specific advice to each group, rather than advising them only on the textbook principles of agile. This approach, following a more *developmental* learning style, increases the relevance of the lessons taken away by each group and—hopefully—allows them to apply the specific advice directly to their project. Students can pull relevant theory as they encounter problems in their project work.

We do not mandate a set development process for the students to follow, but we encourage teams to adopt practices that might be used by an industrial team of a similar size carrying out a similar type of project. We suggest that they follow either Extreme Programming, Scrum, or Kanban (not a mixture)—and back this up with engineering practices such as continuous integration, automated testing and staged deployments. While these practices may not manifest themselves in exactly the same way between different teams, depending on the exact nature of their project, each team should be able to adopt and benefit from most of these in some guise. Again, individual coaching can help teams to adopt appropriate tools and techniques for their specific context.

### 3.6 Checkpoints

As previously explained, we structure these projects as four 2-week iterations. At the end of each iteration project, teams must meet with their supervisor/customer and give them a demonstration of the current state of their software. They should be able to show that they have made progress, and that their software has more (or better) features than it did at the last demo.

To structure these meetings, and to provide some consistency across groups and supervisors, we provide a simple checklist. We want to keep the assessment process lightweight for all parties, so we want to avoid writing and reading detailed reports. The checkpoint forms are quick to complete and quick to check. We provide each team with a 1-page PDF form, which they take with them when they demonstrate their product, and they ask the customer to complete and sign it. The team then scans the signed sheet and submits it as a piece of coursework. We also ask them to submit a set of three screenshots showing the current state of the digital tools they are using to manage three aspects of their project—their version control repository, their continuous integration build, and their project plan. Again, the idea is to have a deliverable that is quick to produce, and quick to check. The checkpoint form has the following questions for the customer to answer:

*I certify that in this iteration I feel the group has... (check one):*

- *not been able to demonstrate any new working software*
- *shown me something working, but a bit less than I had hoped for, or not what we agreed*
- *adequately delivered the features that we agreed on*
- *made better progress than I expected*
- *made amazing progress with wonderful results*

*Has the list of risks to project success changed since you last met the group? What are the two main features agreed to be delivered for the next checkpoint?*

*Customer Signature/Date*

Previously we just had binary checkpoints—either the customer was satisfied or they were not—but we have found that giving a way for customers to express their satisfaction on a scale has led to a more meaningful interaction, as well as more motivation on behalf of the teams to try to please their customers. This system of end-of-iteration checkpoints seems to be working well as a way to produce a more sustained pace of development across the term, rather than a big bang before the deadline, and also provides the benefits of agile development to the customer as they have more opportunities to see the product running, and steer future feature development so that they end up with something that meets their needs.

### **3.7 Summary**

Our main learning in restructuring our project-based courses in Software Engineering was to focus on the regular delivery of working software. Whatever types of technical practice or project management technique we might encourage the students to adopt, they need to feel the benefits of those in helping them to deliver reliable software that meets their customer's needs, without working to a crazy schedule. As we want to foster creativity, and believe that students are more motivated when they get to choose from a wide range of projects, we need to provide support and tutoring that is specific to each team. Following a coaching model allows us to provide relevant, specific, help and advice to each team, at the point that they need it. This seems to be much more effective than more generalised transmission through lectures, especially given the varying needs of the different teams.

By coming up with an overall iteration structure, and a lightweight way of assessment through end-of-iteration checklists, it is possible for us to have a degree of consistency across the class, while still allowing different teams to work in quite different ways. We also found that this outline structure helps students to plan their work across the term, rather than leaving everything until just before the final deadline.

## 4 Future Directions

The methods described in this chapter are working well for us in these Software Engineering modules. But our Computing curriculum comprises many more modules besides these—modules on basic programming, compilers, operating systems, databases, logic, mathematics, etc. It is tempting to try to spread our agile and lean approaches to more modules, but we suspect there will be friction. Just as when introducing agile methods to software development organisations, change is hard. Lecturers running traditional lecture courses may be reluctant to increase the frequency of practical exercises, especially if that means more frequent assessment. The path of least resistance may be to stay with the status quo, but we believe that in transforming the modules described here we have increased their educational value, and hope that we can gradually spread this across our curriculum. Perhaps it is natural that instructors with experience of agile methods are the most keen to introduce them to their teaching, but we hope some enthusiastic colleagues will try to apply similar techniques in their classes too.

Another question is whether we can scale to larger class sizes. Lean methods allow us to improve efficiency and aim to make contact time between students and teachers more valuable. However, the close collaboration and frequent feedback we have implemented in our new structures do not relieve staff time. If we increased class sizes then we would still need more markers, more tutors, more coaches and more customers. Automation helps to remove some trivial tasks and to streamline some of the others, but so far we cannot see a way to remove the human element, and neither are we sure that we would want to.

In fact, the role of the instructor becomes critical. As agile implies high-contact collaboration, working closely together exposes problems and uncovers any lack of experience on behalf of the teacher. When coaching a project team, textbook knowledge is not enough, we need specialists with real experience of running agile projects in the wild. Removing waste from the learning experience has made the need for expert tuition the bottleneck in our system, which rather than a problem, is perhaps to be seen as a sign of success.

**Acknowledgements** We would like to acknowledge colleagues and students at Imperial College London for their contributions to the evolution of our curriculum.

## References

- Anderson, D. (2010). *Kanban: Successful evolutionary change for your technology business*. Blue Hole Press.
- Anslow, C., & Maurer, F. (2015). An experience report at teaching a group based agile software development project course. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education* (pp. 500–505).
- Duvall, P., Matyas, S. M., & Glover, A. (2007). *Continuous integration: Improving software quality and reducing risk*. Addison Wesley.

- Fowler, M. (2011, July). Frequency reduces difficulty [online]. Retrieved from <http://martinfowler.com/bliki/FrequencyReducesDifficulty.html>.
- Gamma, E., Helm, R., Johnson, R., & Glissades, J. (1995). *Design patterns: Elements of reusable object-oriented software*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.
- Guzdial, M. (2015). *Learner-centered design of computing education: Research on computing for everyone*. Morgan & Claypool Publishers.
- Jones, B., Sodhi, R., Murdock, M., Mehra, R., Benko, H., Wilson, A., & Shapira, L. (2014). Roomalive: Magical experiences enabled by scalable, adaptive projector-camera units. In *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology* (pp. 637–644). New York, NY, USA: ACM. <https://doi.org/10.1145/2642918.2647383>.
- Kniberg, H. (2014, March). Spotify engineering culture [online]. Retrieved from <https://labs.spotify.com/2014/03/27/spotify-engineering-culture-part-1/>.
- Kropp, M., & Meier, A. (2014). New sustainable teaching approaches in software engineering education. In *2014 IEEE Global Engineering Education Conference (EDUCON)* (pp. 1019–1022).
- Ottinger, T. (2015). Over-starting and under-finishing [online]. Retrieved October 4, 2016, from <https://www.industriallogic.com/blog/over-starting-and-under-finishing/>.
- Papatheocharous, E., & Andreou, A. S. (2014). Empirical evidence and state of practice of software agile teams. *Journal of Software: Evolution and Process*, 26(9), 855–866. <https://doi.org/10.1002/smr.1664>.
- Rother, M., Shook, J., & Institute, L. E. (2003). *Learning to see: Value stream mapping to add value and eliminate muda*. Productivity Press.
- Williams, L., & Upchurch, R. L. (2001). In support of student pair-programming. In *Proceedings of the Thirty-Second SIGCSE Technical Symposium on Computer Science Education* (pp. 327–331). New York, NY, USA: ACM. <https://doi.org/10.1145/364447.364614>.



# Developing a Spiral Curriculum for Teaching Agile at the National Software Academy



James Osborne, Wendy Ivins and Carl Jones

**Abstract** The National Software Academy (NSA) was established at Cardiff University in October 2016 as a centre of excellence for Applied Software Engineering. We work in partnership with Welsh Government and industry leaders to address the national shortage of software engineering graduates with the skills, knowledge, and hands-on experience required to be immediately effective as commercial software engineers. We run an innovative, industry-focused B.Sc. which uses agile methods to facilitate our project-based learning (PjBL) approach. The projects are provided by our network of industrial partners and are used across multiple modules as a basis for assessment across the disciplines. Although the degree course has yet to produce any graduates, a significant proportion already hold conditional job offers. A conversion M.Sc. in Applied Software Engineering for STEM graduates has also been developed, and will benefit from lessons learned as we continue to inspect and adapt the undergraduate degree programme. This chapter outlines how a spiral curriculum has been developed for teaching agile that progressively introduces complexity whilst building on previous learning.

**Keywords** Spiral learning · Agile project management · Project-based learning  
Industrial collaboration · Applied software engineering

## 1 Introduction

Our Applied Software Engineering degree is developed in collaboration with industry partners in order to ensure that graduates from the programme have the necessary skills to enter industry and be productive from their first day of employment.

---

J. Osborne (✉) · W. Ivins · C. Jones  
Cardiff University, Cardiff, UK  
e-mail: [osbornej8@cardiff.ac.uk](mailto:osbornej8@cardiff.ac.uk)

W. Ivins  
e-mail: [ivinswk@cardiff.ac.uk](mailto:ivinswk@cardiff.ac.uk)

C. Jones  
e-mail: [jonesc162@cardiff.ac.uk](mailto:jonesc162@cardiff.ac.uk)

© Springer Nature Singapore Pte Ltd. 2019  
D. Parsons and K. MacCallum (eds.), *Agile and Lean Concepts for Teaching and Learning*, [https://doi.org/10.1007/978-981-13-2751-3\\_15](https://doi.org/10.1007/978-981-13-2751-3_15)

We work with companies ranging in size from small- to medium-sized enterprises, all the way up to large multinationals, to develop and deliver the curriculum, using a project-based learning approach. Industry interest and support in our approach has been overwhelming. Since our launch in October 2016, over 220 organisations have visited the NSA.

Companies pitch their project ideas to a panel of academics, who review them for suitability of fit within the modules included in the programme. Students work collaboratively in small teams of about four students, developing software for real clients. Students typically spend 30–50% of each semester working on developing solutions for those companies who can take the projects forward as they see fit. The teams adopt an agile approach to development and will meet their clients at regular intervals. Various aspects of the project work form part of the assessment towards their final degree. This project-based approach to learning helps the NSA bring work experience into the teaching environment so that all students benefit from gaining real project experience. All intellectual property is owned by the industrial partner; however, we ask that they allow students to use these projects to showcase their talents to future employers.

Our teaching environment is an open plan office, where students are arranged in small groups of three to four. Students are provided with a laptop, preloaded with the development tools required and can connect to an external monitor, keyboard and mouse as needed. Our teaching sessions usually run for 2.5 h at a time, during which students are taught new concepts for a short time, and then they are given the opportunity to put their learning into practice immediately within the same session. The teaching environment enables collaborative learning during the project phases. Students use physical whiteboards and virtual agile project management tools to gain a shared understanding of the project's progress. Academic staff are on hand to support the teams, but each team is responsible for managing the direction and allocation of tasks for their projects. It is during the project phases that the teaching environment takes on a vibrant 'start-up' atmosphere.

We will reflect on our first 2 years of teaching agile at the NSA. We have been continually improving our approach, whilst taking into consideration feedback from students and industry. We will also reflect on the extent that the teaching approach at the NSA aligns with the values of the Agile Manifesto.

## 2 Related Work

The teaching of Agile Project Management skills in higher education is a well-established practice at both undergraduate (Schilling & Klamma, 2010; Mahnic, 2012; Schroeder, Klarl, Mayer, & Kroiss, 2012; Kropp & Meier, 2013; Anslow & Maurer, 2015; Steghöfer et al., 2016; Matthies, Kowark, & Uflacker, 2016) and postgraduate (Rico & Sayani, 2009; Martin, Anslow, & Johnson, 2017) levels where students typically demonstrate their agile understanding as part of a capstone project,

in contrast to our approach, which builds upon students agile understanding over several taught modules.

Of those sources listed above, only Schilling and Klamma (2010) use real projects set by industry collaborators like we do at the NSA. Unfortunately, their experience was that ‘the establishment of communities of practice between employees of a company and university students was not successful’. We note that successful industry engagement is paramount for the success of our programme. Devedžić and Milenkovic (2011) outline a programme with some similarities to ours, with a wealth of experience gathered over 8 years across both undergraduate and postgraduate courses with a good section on lessons learned, but unfortunately no cooperation with industry collaborators.

Both Mahnic (2012) and Anslow and Maurer (2015) teach agile concepts by sprinting for a full semester using projects set by teaching staff mixing theory and application throughout, whereas Kropp and Meier (2013) adopt an approach similar to ours, covering theory during the first half of the semester and sprinting for the remaining half. In contrast both Mahnic (2012) and Anslow and Maurer (2015) sprint during contact time for their modules, whereas, in our module on Agile Project Management, students sprint during contact time for all modules for the final third of the semester as a mini capstone project.

### 3 Programme Overview

Our undergraduate Applied Software Engineering degree was developed in close collaboration with industry. It is structured as a full-time, 3-year degree programme, with two semesters a year. All modules are 20 credits (200 h) except for the 40-credit Large Team Project in the final year. Project-based learning is embedded throughout the **degree** programme and provides a significant contribution to the assessment in 14 out of the 17 modules. Projects typically run in the last 4 teaching weeks in the semester, except for the Large Team Project that will run across the final semester. Students are encouraged to take optional summer placements at the end of their first and second years (see Table 1).

The majority of assessments are based on students generating a portfolio of examples of work which can be used to showcase their talents to future employers. The quality of portfolio entries is marked against criteria set by module leaders and in accordance with university best practice.

In the first year, students work primarily with languages such as JavaScript, Java and Python to design, develop and deploy mobile and web applications as per the needs of customers. Students learn how to use the same industry standard tools that are used by real-world developers, following best practice to develop quality software. They begin to develop their professional skills, including communication and project management. They work as a development team and gain their first experiences of agile development by applying Scrum as defined in Schwaber and Sutherland (2017).

**Table 1** An overview of the Applied Software Engineering degree programme

Programme overview		
Year 1—Autumn	Year 2—Autumn	Year 3—Autumn
<ul style="list-style-type: none"> <li>• Computational thinking</li> <li>• Introduction to web development</li> <li>• Software development skills 1</li> </ul>	<ul style="list-style-type: none"> <li>• Agile project management</li> <li>• Database systems</li> <li>• Commercial applications with Java</li> </ul>	<ul style="list-style-type: none"> <li>• Commercial frameworks, languages and tools</li> <li>• Adopting technology</li> <li>• Emerging technologies</li> </ul>
Year 1—Spring	Year 2—Spring	Year 3—Spring
<ul style="list-style-type: none"> <li>• Software development skills 2</li> <li>• Fundamentals of computing with Java</li> <li>• Mobile development with Android</li> </ul>	<ul style="list-style-type: none"> <li>• DevOps</li> <li>• Security</li> <li>• Performance and scalability</li> </ul>	<ul style="list-style-type: none"> <li>• Managing software enabled change in large organisations</li> <li>• Large team project</li> </ul>
Year 1—Summer	Year 2—Summer	Year 3—Summer
<ul style="list-style-type: none"> <li>• Optional summer placement</li> </ul>	<ul style="list-style-type: none"> <li>• Optional summer placement</li> </ul>	<ul style="list-style-type: none"> <li>• Graduation</li> </ul>

In the second year, students work on larger, more complex and technically difficult projects. They expand their knowledge in areas such as performance and scalability, databases, security and DevOps. Students are exposed to a wider range of agile approaches and at this point, students are expected to take greater responsibility for their agile practices. Students get the opportunity to serve their teams as Scrum Masters and to work with their customers as Product Owners.

In the final year, students learn about emerging trends and use them to develop a product with an appreciation of the latest frameworks, languages and tools. They take full responsibility for managing and developing a significant client-based project as part of the 40 credit Large Team Project module. The final year builds upon the experiences of years one and two, and brings together all the elements students need to think and work like commercial software engineers. The students learn about the challenges of applying agile practices to large organisations and historical projects since most of their work up to this point will be on new developments.

As staff, we regularly inspect, review and adapt our teaching practices, based on feedback from students and the companies we work with. Sometimes it is possible to make changes mid-semester, however the majority of changes are made in time for the next cohort to begin that stage of the programme. In particular, we would like to thank our first cohort for their feedback as this has been particularly valuable in helping us to adapt the course based on their experience.

## 4 Developing Spiral Learning for Teaching Agile

This section reflects on our first 2 years' experience in teaching agile.

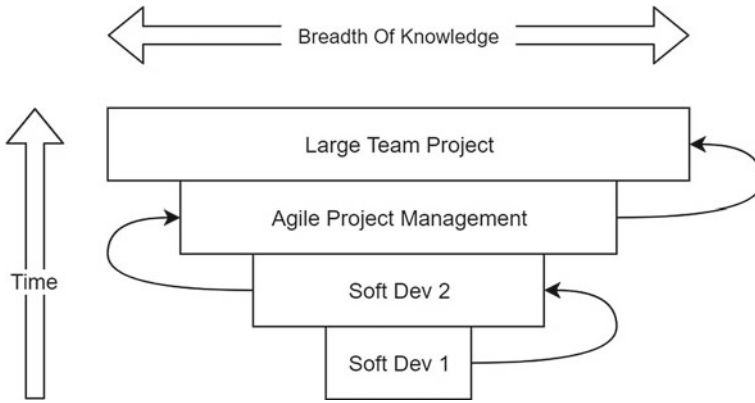
Pedagogically, the university adopts an approach to programme development called constructive alignment (Biggs & Tang, 2011) which involves developing teaching and learning activities, and assessment tasks, that assess well-defined learning outcomes. Those learning outcomes are set at the programme and module level. Learning outcomes and assessments are reviewed periodically to make sure they are up to date, and teaching and learning activities are also updated in line with those learning outcomes. The current learning outcomes for the Agile Project Management module in the 2nd year of our degree programme are as follows:

1. Apply agile project management principles to the development of user stories, the breaking down of stories into tasks, and the management of project and sprint backlogs in collaboration with an external product owner using the Scrum and/or Kanban framework.
2. Apply agile project management principles both whilst leading (as scrum master) and taking part (as scrum team member) in planning, daily scrum, and retrospective meetings with the rest of their group using the Scrum and/or Kanban framework.
3. Apply agile project management principles whilst taking part in product review meetings with their external product owner and potentially other project stakeholders.
4. Compare and contrast Scrum and Kanban frameworks.
5. Reflect on their application of agile principles to a real-world software project.

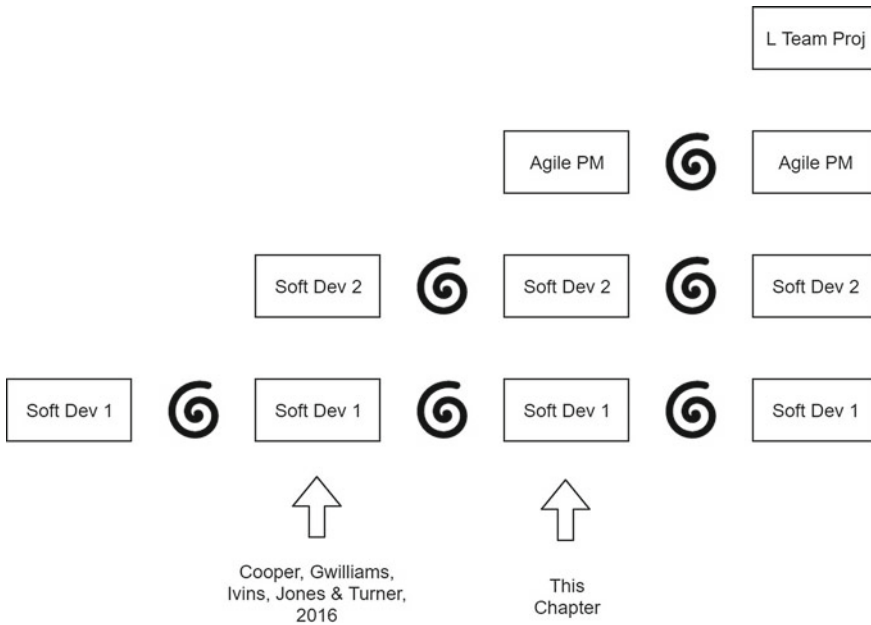
We also use an approach called Spiral Learning or Spiral Curriculum (Bruner, 1960), where students build up their understanding of a topic, in this case, Agile Project Management, by repeated exposure to the concepts, starting with the core concepts, and building upon this foundation with more complex concepts, to reinforce prior learning (see Fig. 1). Spiral learning can be further broken down into two types, horizontal and vertical integration (Brauer & Ferguson, 2014).

Horizontal integration works across modules delivered in a single semester. Client-based projects span these modules so lecturers work in semester teams and collaborate to schedule and deliver teaching, coordinate and scope projects, whilst ensuring all learning outcomes are addressed. Our horizontal integration is similar to the Spotify model (Kniberg & Ivarsson, 2012) where semester teams resemble a single squad. We use the term 'semester team' as this is terminology more familiar to the wider academic community than a squad where, for example, the Agile Project Management module leader reviews the process, the Database Systems module leader reviews the database layer, and the Commercial Applications with Java module leader reviews the Spring Boot application itself.

Vertical integration represents integration across time (see Fig. 2). Thematic teams, such as the agile development team, collaborate and share experiences to ensure effective scaffolding across the programme. Our vertical integration is again



**Fig. 1** An overview of Spiral Learning showing how breadth of knowledge increases over time as students progress through the modules



**Fig. 2** An overview of Spiral Learning showing how modules are refined over time as we continue to inspect and adapt the programme with reference to a previous research output and this chapter

similar to the Spotify model (Kniberg & Ivarsson, 2012) where thematic teams resemble chapters. Again, we use the term ‘thematic team’ as this is terminology more familiar to the wider academic community than chapters. The agile spiral begins in the first year as part of two modules on Software Development Skills. In the second

year, students complete both the Agile Project Management and DevOps modules, and in the final year work as part of a Large Team Project.

## **4.1 *Our First Year—2015–16***

### **4.1.1 First Cohort—Year 1—Autumn Semester 2015**

The client-based projects ran in the last 4 weeks. Those projects were web applications for: (a) the School of Computer Science, (b) a Local Dental Practice, and (c) the Welsh Football Association. Two teams of 3–4 students worked on each project.

During the semester, we introduced students to a minimum of agile practices as part of the Software Development Skills 1 module. We assumed students had no prior knowledge of coding or working as part of a team. We asked the students to begin by developing skills in self-learning, knowledge sharing, problem-solving techniques, basics of code quality and the simplest use of source control before introducing the students to their commercial customers.

Students were taught how to ask good questions during client meetings, how to record notes and playback understanding to develop requirements that were of value to the user (e.g. User Stories), how to formulate an initial backlog of requirements and how to track development progress. We briefly discussed Scrum as a methodology allowing students to use as much or as little as they could learn independently for their projects. We also encouraged the use of Trello (2018) and Slack (2018) for project communication.

Our main aim was to teach the students just enough agile to perform and to encourage them to learn from their mistakes. In terms of Scrum process over the 4-week project period, students were able to complete two 2-week iterations, the first of which began with a client meeting to gather requirements, followed by a show and tell session at the project mid and end points with the client, followed by a team retrospective.

We were able to compare team behaviours and dynamics as well as progress against user stories and application of agile methodologies across all teams. Students received feedback as they progressed from academic staff, which supplemented feedback from the industrial client during review meetings.

### **4.1.2 First Cohort—Year 1—Spring Semester 2016**

The projects ran over the last 6 weeks of the semester across all modules. Those projects were mobile applications using Bluetooth beacons from a company called GCell for: (a) GCell, (b) a Local Historian, and (c) Newport City Council. Again, two teams of 3–4 students worked on each project.

During the semester, we expanded upon student's understanding of agile practices as part of Software Development Skills 2. We taught students further use of source

control techniques such as branching, and how branching strategies can be used to maintain separate development and master branches. We also discussed feature branching, hinting at the appropriateness of that branching strategy for the DevOps module in the second year.

We briefly discussed estimation using T-Shirt sizing as well as the idea of *#NoEstimates*. We also introduced the concepts of Shu-Ha-Ri, well-formed user stories, Test Driven Development, Behaviour Driven Development and an alternative project tracking tool (Taiga, 2018). We aimed to cover refactoring with students at this time, however, since the students had only just been exposed to Object Oriented Development, they were unable to appreciate the need for refactoring, given the maturity and complexity of the code bases they were working on.

Our aim was that by the end of the first year, students should understand the principle of delivering value to the customer at regular intervals, having software in a deployable state, team communication and measurement, and using those measurements to inform next decisions.

The reality was, however, a little different. Students grasped Scrum and User Stories well enough and Taiga was preferred over Trello by most. The ethos of *#NoEstimates* was completely missed by most, and performing estimation ground to a halt, since estimation was not actually being assessed. This was something we would be able to consider in the second year.

Overall, students found it difficult to link the separate concepts, often resulting in stories that were too big, branches that lasted too long, making merging difficult and deployments impossible. The teams were able to demonstrate working software and the clients found value in the student's work. However, the students had not applied agile practices as well as we hoped. Therefore, it was decided to introduce agile more formally in the first semester of the first year.

The lack of discipline at this stage was not such a total disaster. Some students could reflect on their mistakes, though not as many as we would have liked. Again, this would be something to refine in the second year, as valuable learning comes from reflecting on these experiences.

## ***4.2 Our Second Year—2016–17***

### **4.2.1 Second Cohort—Year 1—Autumn Semester 2016**

The second time we ran the semester, we made several changes. We again reserved the last 4 weeks across all modules for the students to work on their commercial projects. This time, our biggest challenge was scalability, as we had increased our intake of students by a factor of three. Again, the commercial projects were web applications for: (a) an Independent Financial Advisor, (b) our Local Dental Practice and (c) the Newport Business Improvement District. This time around, five teams of 4–5 students worked on each project.



This time we decided that students should use the Scrum framework (Schwaber & Sutherland, 2017) more formally to help manage their commercial projects. Clients would meet with the students at the start of the project to discuss their needs. The students would then demonstrate their progress to their clients in a show and tell session after 2 weeks, and the students would provide a final demonstration to their clients at the end of 4 weeks, as before.

Teams were expected to run weekly Sprint Planning sessions to determine the User Stories they would implement in the week's sprint, then break them down into suitable tasks. Students were asked to ensure that the User Stories provided value for the client and were introduced to Cohn's (2004) template for writing user stories: 'As a <type of user>, I want <some goal> so that <some reason>.' Students were also required to develop acceptance criteria for each User Story.

Students were expected to hold a Daily Scrum Meeting when they had project time allocated in their timetable. During the sprint, the teams were expected to use other good practices to manage their software, such as source control and code review.

Each team would meet with an academic staff member every week for a Sprint Review, to discuss their progress on the project in the previous week and to talk through their plans for the subsequent sprint. Academic staff took on the role of Scrum Master to provide guidance on the use of Scrum. Academic staff saw improvements in the planning of projects over the 4 weeks.

Students initially struggled to write user stories that provided value to the client and produced stories that were too technically focused such as 'Produce Database'. They also struggled to reduce the stories into suitable tasks. Later in the project, the students produced more suitable user stories with clear acceptance criteria and broke these down into suitable tasks.

The projects also needed to meet the learning outcomes for the modules. The students were assessed against their ability to set appropriate User Stories and acceptance criteria, as well as to plan and track their project using Scrum. Assessment was also made on the student's reflection on their use of the Scrum framework to 'address complex adaptive problems, while productively and creatively delivering products of the highest possible value' (Schwaber & Sutherland, 2017).

The students had undergone a steep learning curve across all three of their modules, and we did not want this first introduction to agile to be too complex. Therefore, we left some key concepts to be covered in Software Development Skills 2 in the Spring Semester:

- We did not cover estimation and tracking the progress of the project with Burndown charts. Students were asked only to reduce their User Stories if they could not be completed in a 1-week sprint.
- We did not fully implement the Definition of Done, because we did not have clients as Product Owners in the Sprint Planning, Review, Daily Scrum and Retrospective Meetings. Academic staff would discuss the progress of the project and plans for the subsequent Sprint with each team during the weekly Sprint Review Meeting.

Students got an opportunity to demonstrate their software to their client at the midpoint of the project, and upon completion of the project after 4 weeks. Students

were encouraged to demonstrate working software in each of these meetings as ‘Working software is the primary measure of progress’ (Agile Manifesto, 2001). All team members were advised to demonstrate their own work and practice their demonstration before the meeting.

Students demonstrated their software and received feedback from the client in front of the other four teams completing the same project for the client. All 15 teams demonstrated working software at both meetings. The clients were pleased with the progress of most of the teams and each client gained a range of valuable ideas for their proposed project. Several of the teams made impressive progress, given that they only had 4 weeks to work on the projects and each client had at least one outstanding project.

#### **4.2.2 Second Cohort—Year 1—Spring Semester 2017**

In the second iteration, the emphasis shifted from the agile learning outcomes to design-based learning outcomes. Also, with larger numbers of students, a build-up of technical debt could reduce the productivity of many teams and go beyond the capacity of the teaching team to support. It was also felt that the agile principles had been covered in greater depth in the autumn semester and so there was scope to do less in the spring semester. The module still took time to introduce estimation and requirements decomposition (via User Story splitting) in order to get the students thinking about predictability. It also introduced Git branches and source control workflows linking these to story splitting, as a way of reinforcing the idea of frequent delivery of value and the developers’ responsibility for code quality in maintaining a flow of value.

We observed that the students slightly improved their code, design and testing, but they didn’t partition their work successfully. Too often, a particular student would only implement part of a feature rather than the full end-to-end slice. This not only affected their ability to learn the complete development stack but also led to a number of partial solutions being committed and the improvement in design and discipline was not sufficient to deal with this change of workflow.

With these two forces in mind, it will be necessary to find the right balance between writing the right software and writing the software right. The pendulum has swung both ways, and we need to re-emphasise the decomposition of work into features and the focus on quality to enable smooth integration of those features into the deliverable.

#### **4.2.3 First Cohort—Year 2—Autumn Semester 2016**

This was the first time we ran our year 2 modules. We reserved the last 6 weeks across all modules for the students to work on their commercial projects. Those projects were multi-tiered applications: (a) A Wellness Monitoring System for COPD Sufferers for Medivate, (b) A Performance Monitoring System for Grassroots Players for the

Welsh Rugby Union and (c) A Back-Office System for Lettings and Maintenance for the Zest Letting Agency. One team of 7–8 students worked on each project. During this iteration, the Agile and DevOps modules were taught concurrently across both semesters rather than consecutively.

Students took on the role of Scrum Master and convened meetings as appropriate. Students also had to familiarise their industrial partner with agile methodologies, with as little assistance as possible from lecturing staff, so their industrial partner could act as a Product Owner.

Scrum Masters would work together with Product Owners to define the Product and Scrum Backlogs. Agile topics that we expanded upon in the first semester included: Plan Do Check Act, Scrum Team Dynamics, Lean Start-up, Information Radiators, Collective Ownership, Working Agreements, Pair Programming, Refactoring, Product Owners, Scrum Masters, Developers, Testers, Sprint Planning, Planning Poker, Backlogs, Features, Flippers, INVEST Stories, SMART Tasks, Spikes, Bugs, Definition of Done, Scrum and Kanban boards, WIP Limits, Daily Scrum and Retrospectives. Based on our industry experience, and the topics we wanted the students to understand, some of which are listed above, our recommended reading included approximately 75% of the material from Cohn (2004), parts II and III on individuals and teams from Cohn (2010), and approximately 66% of the material from Rubin (2012). Although the lecture materials sufficiently covered those topics, having exposure to Cohn and Rubin as thought leaders, with a different way of explaining topics, with their own experiences and case studies has enriched the students' learning experience.

The Scrum artefacts required for the project included: Product and Sprint Backlogs from Sprint Planning Meetings with stories that are INVEST (Independent, Negotiable, Valuable, Estimable, Small and Testable) and tasks that are SMART (Specific, Measurable, Achievable, Relevant and Time-Boxed), that have been estimated using Planning Poker; Brief notes from Daily Stand Up Meetings about Stories, Tasks, Blockers and Spikes and how they were being resolved or investigated; Customer comments and assessment of professionalism at Sprint Review Meetings where the demonstration of working software was paramount; and techniques used in Sprint Retrospective Meetings, such as those presented in Derby and Larsen (2006).

We were also interested in the students presenting snapshots of stories at regular intervals so that we could monitor quality and estimation practices. For example, using a Burndown chart to estimate Velocity for subsequent Sprints, to set Sprint Scope and Sprint Goals, and of course the use of either electronic or physical Scrum Boards, optionally with WIP (Work In Progress) limits, if the teams were applying some of the principles of Kanban.

We would like to take this opportunity to thank the members of the South Wales Agile Group (2018) for running several activities with the students including (a) getKanban, a game which simulates a development team working on a project using the principles of Kanban (The getKanban Board Game, 2018); and (b) the Lego Flow

Game, a game which uses the Star Wars advent calendar to compare Waterfall, Scrum and Kanban processes as applied to Lego Minifigure construction (AvailAgility, 2018).

#### **4.2.4 First Cohort—Year 2—Spring Semester 2017**

The focus in this semester changed from the development of functionality to meet client needs to addressing the technical debt in order to improve deployability, security, performance and scalability. This is the only semester when clients were not directly involved. The semester team were concerned that if we continued to assess the students as a group then not all students would get the same opportunities to meet all of the learning outcomes. The students were therefore assessed individually in this semester.

We expanded on students understanding of agile principles by introducing Scaled Agile Framework, Large Scale Scrum, and Disciplined Agile Delivery; alongside other agile methodologies including Extreme Programming, Crystal, Dynamic Systems Development Method and Feature Driven Development; as well as more traditional methodologies for software development to understand how agile methodologies have evolved and that agile methods may not be appropriate for all software engineering projects.

We also introduced the students to a number of case studies from a range of public and private companies showing their use of agile practices and how well those practices aligned with the Agile Manifesto. The assessment for this module was a portfolio entry worth 80% of the marks for the module. This assessed the topics mentioned above, as well as their reasoning as to why they may or may not want to work for the companies we reviewed giving reasons.

In terms of the DevOps module, students were asked to create a Terraform plan (Terraform, 2018) that could be run by a member of staff, to deploy the complete solution for their commercial customer into our departmental cloud. The staff member would be able to see tests run and interact with the system to retrieve data from the database layer, through to the web presentation layer. The assessment for the DevOps module was a portfolio entry assessing the quality of their deployment plans. The remainder of the marks came from a timed exercise, deploying database X to server Y by developing a Vagrant script (Vagrant, 2018) and an overview of how the students could implement continuous integration and deployment pipelines into their development processes.

## **5 Summer Placements**

At the time of writing, our first cohort of students is halfway through the first semester of their final year, meaning we have yet to produce any graduates from the programme. We encourage companies to offer summer placements as a means of giving

an extended interview to our future graduates. Companies use their own selection processes to choose students, including technical tests and interviews. In return, students gain experience of a range of interview techniques and have the opportunity to work in those companies to see if they fit both technically and culturally into their environment.

Our current understanding of the work-readiness of students has been reinforced by the positive feedback from the employers. A number of students from our first cohort are already holding conditional job offers which validates both the programme and the project-based learning approach for us.

### ***5.1 First Cohort—Summer 2016***

Our first batch of Summer placements were offered to our current third year students. Five students went on placement with four industrial partners that we had talked to while setting up the programme. Feedback from the industrial partners was overwhelmingly positive. Some even highlighted that the companies benefitted from the student's use of agile practices.

“StudentA joined our web team at the start of the summer and immediately got to work helping us with the project delivery” ... StudentA started with some small bug fixes, things like wording or layout changes and then progressed to more complex business functionality. StudentA is a quick learner, enthusiastic and knowledgeable. StudentA got on with the team and those around them, and took an active role in team meetings and daily scrums. ... StudentA demonstrated their progress during the build of the tool to the team through our regular show and tell sessions, in which they were confident and communicated well.”

“CompanyB has supported the innovative and industry-need based approach that underpins the philosophy of the NSA since its inception.” ... “This summer, we had the opportunity to experience, first-hand, the benefits of the approach, from an employer's perspective, by offering two 12-week summer placements. It was an entirely positive experience. Their familiarity with the agile processes we use and the skills they had gained in extending their knowledge and skills quickly to meet new challenges meant that within the first 2 weeks it felt as if they were long-standing valued, members of the team.”

### ***5.2 First and Second Cohorts—Summer 2017***

Our second batch of Summer placements were offered to our current second and third year students. Eighteen students went on placement with twelve of our industrial partners. Again, they were extremely successful and a number of them already have conditional job offers from the companies since their placement. Here are some comments from those companies about those students with conditional job offers.

StudentB has been fantastic addition to the team, despite him being new, he was not afraid to voice his opinion, particularly on how to implement agile and DevOps methodologies correctly. His course has taught him all this and this is something very new to us. His

programming knowledge is also very good, he is currently working on rewriting the 360 degree feedback site using Spring Boot, which is coming along quite nicely. He has built a prototype quite quickly, considering he has had other work to do such as helping our SME write stories for sprints. He has been very quick to pick up new things and it feels as though he has always been there, it does not feel like he is an intern.

StudentC has been a blessing to the team these last few months. She is always willing and eager to learn and will jump into help with anything you need, she has picked up the work we do within data support really quickly and has suggested ideas on how to improve our processes. She has really helped us over these last few months, especially picking up the best practice document that has been so well written and taking it out to the business and getting their buy in. She has fitted in with the team well and has really taken on board the CompanyC spirit and is always extremely helpful. Overall I think she has been great asset to the team and will be missed.

## 6 Reflection

### 6.1 *Pedagogical Constraints on Teaching Agile*

First and foremost, the NSA is part of a larger educational institution with processes around curriculum design and delivery that influence heavily the way in which we can teach agile practices to our students. Our university prefers all academic staff to use constructive alignment (Biggs & Tang, 2011) to define a series of learning outcomes for each module.

The NSA uses a project-based learning approach as its main way of delivering many of the learning outcomes in the programme (Cooper, Gwilliams, Ivins, Jones, & Turner, 2016). It is therefore vitally important that industry projects are carefully selected to deliver against the relevant learning outcomes of each of the modules delivered in a semester. Students are assessed primarily through project portfolios that contain project deliverables, samples of work, and reflection on project learning.

We are in the fortunate position of being able to select from a wide range of industry projects. However, if any industrial projects are unable to deliver against all the learning outcomes, we have developed some internal projects that can be used instead.

We are also fortunate, though some may argue against this, that as a staff we have a single open plan office at the NSA which is a great way to foster collaboration and to allow other members of the team to learn agile practices from their peers. This became vital as staff, not experts in agile methods, became involved in the assessment process for the first year students and were involved in Sprint Reviews. Students benefitted from this weekly formal feedback so regular reviews with academic staff will be adopted across all client-based projects in the future.

Assessments typically flow from artefacts that students are generating as part of their application of agile methods, to their industry projects. Students keep a portfolio including a wide range of items, the quality of which can be easily assessed. Students

are also asked to write reflective essays summarising the positive and negative aspects of their industrial projects.

Industry partners are also encouraged to mark the students on their level of professionalism and engagement, though there must be an element of moderation led by staff. Clients marked against a set of assessment criteria for the first year students but they did not provide marks. We must also ensure that assessments work across modules in combination so as students are not overwhelmed with assessments for modules all due in quick succession at the end of the semester.

A key consideration for effective team working is setting a team size that is right for the context of the project. Projects in the first year are relatively simple and are delivered in a short timeframe of typically 4–6 weeks. Team sizes should be small (ideally 4–5 students) to ensure that each student can make a fair contribution to the project. We learned that teams of 3 were simply too small and decided that for the second year, we would have larger teams of 7–8 students instead. However, as team sizes grow, then it is easier for students to hide behind the achievements of other team members, unless there is a regular and effective way of monitoring each student's contribution. The weekly reviews with staff used with the second cohort of first years were found to be an effective way of monitoring each student's contribution to the project, and regular reviews with staff will be used in all client-based projects in the future.

The remainder of this section reflects on how we promote the core values of the Agile Manifesto and the value of an agile approach to all the stakeholders.

## ***6.2 Individuals and Interactions over Processes and Tools***

Students have timetabled contact sessions for 15 h a week but are expected to work a 40-h week, completing a significant amount of project work outside of scheduled sessions. This means that teams work as a co-located team during contact sessions, and a distributed team outside of timetabled hours. This presents real challenges for interactions outside of contact sessions and maintaining awareness of what the other team members are working on.

The first cohort of students, currently in their final year, were using a combination of both physical and electronic Scrum/Kanban boards. Though students are working in a regular Scrum cadence, some of them understand the principles of WIP limits and have decided to apply them to their projects. This is possible because there are only 20 students working across three projects and there are enough physical whiteboards to go around.

The problem arises when they are working off-site or in distributed fashion, and collaborative tools such as Trello and Slack and even Facebook Messenger come into play. Students take on the role of Scrum Master in the second year, a role which rotates around all team members on a per Scrum basis, and they are responsible for convening the Sprint Planning, Review, Daily Scrum and Retrospective Meetings. Staff are on hand to advise during Planning, Review, and Retrospective Meetings.

The second cohort of students, currently in their second year, could maintain a shared awareness of the project progress through online tools such as the Scrum Template in Taiga, and collaborative tools such as Microsoft Teams (Microsoft, 2018). It was impractical to provide sufficient whiteboards in our physical space for 15 teams. It was also much easier for academic staff to discuss the effectiveness of planning and tracking the projects in the weekly Scrum Review Meetings, as the Scrum Template became a key artefact to centre the discussion around. Academic staff acted as Scrum Masters and gave targeted advice about each team's use of Scrum in the weekly Scrum Review Meetings. Students could also seek advice on their use of Scrum during their project sessions.

Students were introduced to version control with Git in the first week of the first semester using a Gitlab repository (Gitlab, 2018). Their use of Git at this time is rather unsophisticated as the concept of branching and merging is only being taught during the second semester of the first year. As students progress to the second year, Git is also used to manage infrastructure as code as part of the DevOps module, which initially ran concurrently with the Agile Project Management module in the second year. Some of the students also maintain their own remote repository using GitHub (Github, 2018), though they are encouraged to use school services where possible. GitHub has effective inspection tools to show when students are committing code to the project, helping staff determine to what extent team members are contributing to the project. These tools must be used with caution as they do not indicate the amount and quality of the source code that has been contributed.

As students progress through the programme, their understanding of the advantages and disadvantages of working as part of a team develops. Students are often overly concerned with how they are being marked when working as part of a team. Some of them are more interested in making sure they do well in the assessment to the detriment of contributing to the team in a positive way. Addressing this issue is something we continue to struggle with.

Additionally, we have had to step in several times when team members have been unable to get along with each other. Some of us believe we should encourage the students to work through their differences, as would be the case in the real world. However, if the team was unable to resolve these issues, then academic staff have stepped in and moved team members onto other projects.

### ***6.3 Working Software over Comprehensive Documentation***

Delivery of working software is our clients' preferred measure of success. Students need to be seen to be delivering features on their projects, and to show progression of software incrementally at each Sprint Review Meeting. When working as part of a team, students are encouraged to commit early and often, to prevent merge conflicts and enable Continuous Integration and Continuous Deployment.

We have taken on the mantra of 'if it isn't in the repo, it doesn't exist' as a way of dealing with student excuses for not completing work. We would like the students to



take that to heart, making commitments to each other, so that when they say they will deliver a piece of functionality by a certain time, that they will do so. Some students are however more diligent than others—as are developers in the real world—and sometimes students find it difficult to balance delivering working software, with working on their assessments.

Teams of students also informally discuss their projects with their peers, allowing ideas to cross-pollinate between projects. If more than one team is working with a customer, it allows different teams to work on different Product and Sprint Backlogs independently so that different functional themes may be explored. Moving forward, it might be nice to use a more formal Scrum-of-Scrums process with students to encourage further cross-pollination.

Given our experience of working on industrial projects, we are a little concerned with the quality of documentation produced by the students, all the way from documenting architectural and platform decisions, through to documenting Scrum Meeting outputs. Their use of Scrum and Kanban boards has been impressive, facilitated by readily available tools for them to use. We have not standardised on a Wiki platform or similar, something we should probably do sooner rather than later, to facilitate document sharing and storage for students.

Working in a university environment, staff are regularly working on documentation which, for procedural reasons may have to be stored in several different places including learning central—the online system we use to store lecture materials and manage assessments—and shared drives for documents pertaining to programme management and assessment.

Some of the processes within the university that we must adhere to certainly cannot be described as agile, however we use agile principles to adapt the curriculum in response to feedback from industrial partners and students as soon as possible, feeding adaptations into the university module review processes as soon as we can.

## ***6.4 Customer Collaboration over Contract Negotiation***

Customer collaboration comes mainly through the client projects. Students are engaged because they have real-world projects and they can see the value of their work through the feedback from the clients. However, there is a limitation to the amount of time the clients can give to the project, making it difficult for the clients to fully act as Product Owners.

This is a challenge with the growth of student numbers in our second cohort, as the client is giving feedback on five independent projects. This meant that each team took responsibility for prioritising the Product Backlog Items based on their understanding of the client's needs and selected the items for each Sprint. One advantage to the client is that they get a range of ideas from the different teams. These teams also sit in on the other teams' demonstrations and listen to the client feedback, which students found to be insightful.

Client-based projects are the main catalyst for learning at the NSA (Cooper, Gwilliams, Ivins, Jones, & Turner, 2016) but these projects must allow students to achieve their learning outcomes, across the appropriate modules, through the execution of the project. Clients gain value through the ideas presented by the different teams.

Students gain important softer skills through their interaction with their customers. They must elicit the requirements from their clients to develop appropriate user stories and acceptance criteria. They develop their communication skills through regular demonstrations and interactions with different clients during their degree programme.

A key challenge is maintaining a healthy pipeline of suitable projects across the programme. Considerable time is needed to establish the relationships with our industrial partners, scope the projects and manage client expectations. This is a key responsibility of the School Manager at the NSA, who spends considerable time working on stakeholder engagement. In total, we have been visited by over 220 different organisations. We have also been visited by the Welsh Government Minister of Economy, Science and Transport, and the UK Government's Director General, Business and Science, Department for Business Innovation and Skills. We have also gained industry recognition to win the ESTNet Collaborative Partnership of the Year Award 2016 (ESTNet, 2016) and the ESTNet Industry Trailblazer of the Year Award 2017 (ESTNet, 2017).

Working with industrial clients to develop prototype or proof-of-concept systems often requires that staff and students at the NSA enter into non-disclosure agreements with customers that may wish to protect an element of intellectual property in the software system developed. Normally, the university would retain intellectual property rights of anything developed by staff and students during their time at university—the same often being said of developers working on side projects whilst working for a company—however the NSA is different in this regard. We have our own form of Non-Disclosure Agreement in which the university and students waive any intellectual property rights, and that the software system developed is only a proof-of-concept. If the customer wishes to develop the system for commercial gain, they must deploy the code on their own systems.

## ***6.5 Responding to Change over Following a Plan***

Students are taught that they should spend less time planning and more time developing and evolving systems, as is the agile way. We know it is impossible to predict with any certainty beyond a given time horizon and students come to appreciate the wisdom of this approach early on. The Sprint Planning Meetings they hold do involve Backlog Grooming, making sure stories are well formed and that estimates are derived as appropriate and metrics are captured.

Given the light-touch we use for planning, students find it easier to adapt to change and even expect it, although they may not be able to come up with mitigating actions,

as plans change during the first year. Hopefully, with experience and wisdom, their ability to plan for change will improve.

Students in the first year benefit from regular feedback by reviewing plans with staff on a weekly basis. The first year students were introduced the concept of the Sprint Retrospective rather late in the cycle, as we wanted them to concentrate on learning how to apply other aspects of agile practice first. The first years were not expected to come up with estimates of effort beyond making sure that user stories in each Sprint could be completed within a week. Towards the end of the semester, each team held a retrospective to review progress and teams were encouraged to continue the practice into the next semester.

Students were then introduced to estimation in the second semester and started using the process in their client projects. In the second year, we go into greater depth with estimation using Planning Poker as part of the Sprint Planning Meetings, subsequently used to calculate Velocity and therefore, manage customer expectations on what will be delivered during a Sprint. This may either last for 1 or 2 weeks, depending on the cadence teams can use with the client.

For the university, we must fit into the programme approval process, which means that, depending on the scope of changes we make, we may have to work through either a light-touch change management process for minor changes, or a more rigorous change process for more significant changes.

We have recently completed the university approvals process for a new conversion Masters level qualification, aimed at graduates from STEM programmes, and we look forward to delivering this qualification for the first time in the 2018–19 academic year.

## 7 Conclusions

Working within the constraints of academia whilst teaching agile practices to students, provides us with a constantly evolving challenge. From an academic perspective, we want students to think critically about everything they do and how they apply agile practices to their group projects. We don't just want the students to go through the motions of following an agile process, we want them to understand agile in such a way that they can both apply and teach agile practices wherever they end up, so they are disruptive in a positive way on entering the workforce.

We appreciate the enormous value that project-based learning has brought to the programme in developing work-ready graduates. Students are given a chance to put their agile learning into practice, in a safe environment, with real customers. Using agile concentrates the students on delivering working software and managing projects within a sensible and lightweight framework.

With the arrival of the second cohort of students, we realised that we needed to introduce agile principles even earlier than with the first cohort.

As academics, we must balance the time we spend teaching agile concepts, with allowing students the time to run projects and hold project meetings during regu-

lar contact hours. This remains one of the key challenges to teaching agile in the structured way required in academia with a semi-structured timetable and a need to assess students understanding of the subject. Our approach to spiral learning allows students to learn incrementally and progressively develop their agile practices. This allows us to break down what are truly artificial barriers between modules that can be problematic, in a more traditionally structured modular degree programme.

## References

- Agile Manifesto. (2001). Manifesto for agile software development. Retrieved from <http://agilemanifesto.org/>.
- Anslow, C., & Maurer, F. (2015). An experience report at teaching a group based agile software development project course. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education—SIGCSE '15*. <http://dx.doi.org/10.1145/2676723.2677284>.
- AvailAgility. (2018). Lego flow game. Retrieved from <https://availagility.co.uk/resources/games/lego-flow-game/>.
- Biggs, J., & Tang, C. (2011). *Teaching for quality learning at university*. Open University Press.
- Brauer, D., & Ferguson, K. (2014). The integrated curriculum in medical education: AMEE Guide No. 96. *Medical Teacher*, 37(4), 312–322. <https://doi.org/10.3109/0142159x.2014.970998>.
- Bruner, J. (1960). *The process of education*. Harvard University Press.
- Cohn, M. (2004). *User stories applied: For agile software development*. Addison-Wesley.
- Cohn, M. (2010). *Succeeding with agile: Software development using scrum*. Addison-Wesley.
- Cooper, I., Gwilliams, C., Ivins, W., Jones, C., & Turner, M. (2016). Developing work-ready software engineers using real-world team-based projects as a catalyst for learning. In *7th Annual International Conference on Computer Science Education: Innovation & Technology (CSEIT 2016)*. [http://dx.doi.org/10.5176/2251-2195\\_cseit16.37](http://dx.doi.org/10.5176/2251-2195_cseit16.37).
- Derby, E., & Larsen, D. (2006). *Agile retrospectives: Making good teams great*. The Pragmatic Bookshelf.
- Devedžić, V., & Milenković, S. (2011). Teaching agile software development: A case study. *IEEE Transactions on Education*, 54(2), 273–278. <https://doi.org/10.1109/te.2010.2052104>.
- ESTNet. (2016). Collaborative partnership of the year award. Retrieved from <https://www.estnetawards.co.uk/estnet-award-winners-and-finalists-2016/>.
- ESTNet. (2017). Industry trailblazer of the year award. Retrieved from <https://estnetawards.co.uk/winners-finalists-2017/>.
- Github. (2018). Built for developers. Retrieved from <https://github.com/>.
- Gitlab. (2018). Complete DevOps. Retrieved from <https://about.gitlab.com>.
- Kniberg, H., & Ivarsson, A. (2012). Scaling agile @ Spotify with tribes, squads, chapters & guilds. Retrieved from <https://blog.crisp.se/wp-content/uploads/2012/11/SpotifyScaling.pdf>.
- Kropp, M., & Meier, A. (2013). Teaching agile software development at university level: Values, management, and craftsmanship. In *2013 26th International Conference on Software Engineering Education and Training (CSEE&T)*. <http://dx.doi.org/10.1109/cseet.2013.6595249>.
- Mahnic, V. (2012). A capstone course on agile software development using scrum. *IEEE Transactions on Education*, 55(1), 99–106. <https://doi.org/10.1109/te.2011.2142311>.
- Martin, A., Anslow, C., & Johnson, D. (2017). Teaching agile methods to software engineering professionals: 10 years, 1000 release plans. *Lecture Notes in Business Information Processing*, 151–166. [http://dx.doi.org/10.1007/978-3-319-57633-6\\_10](http://dx.doi.org/10.1007/978-3-319-57633-6_10).
- Matthies, C., Kowark, T., & Uflacker, M. (2016). Teaching agile the agile way—Employing self-organizing teams in a university software engineering course. In *2016 ASEE International Forum*. <https://peer.asee.org/27259>.

- Microsoft. (2018). Microsoft teams. Retrieved from <https://products.office.com/en-us/microsoft-teams/group-chat-software/>.
- Rico, D., & Sayani, H. (2009). Use of agile methods in software engineering education. In *2009 Agile Conference*. <http://dx.doi.org/10.1109/agile.2009.13>.
- Rubin, K. (2012). *Essential scrum: A practical guide to the most popular agile process*. Addison-Wesley.
- Schilling, J., & Klamma, R. (2010). The difficult bridge between university and industry: A case study in computer science teaching. *Assessment & Evaluation in Higher Education*, 35(4), 367–380. <https://doi.org/10.1080/02602930902795893>.
- Schroeder, A., Klarl, A., Mayer, P., & Kroiss, C. (2012). Teaching agile software development through lab courses. In *Proceedings OfThe 2012 IEEE Global Engineering Education Conference (EDUCON)*. <http://dx.doi.org/10.1109/educon.2012.6201194>.
- Schwaber, K., & Sutherland, J. (2017). The scrum guide. Retrieved from <https://www.scrumguides.org/docs/scrumguide/v2017/2017-Scrum-Guide-US.pdf>.
- Slack. (2018). Where work happens. Retrieved from <https://slack.com/>.
- South Wales Agile Group. (2018). South wales agile group. Retrieved from <https://www.meetup.com/en-AU/South-Wales-Agile-Group/>.
- Steghöfer, J., Knauss, E., Alégroth, E., Hammouda, I., Burden, H., & Ericsson, M. (2016). Teaching Agile. In *Proceedings of the 38th International Conference on Software Engineering Companion—ICSE '16*. <http://dx.doi.org/10.1145/2889160.2889181>.
- Taiga. (2018). Love your project. Retrieved from <https://taiga.io/>.
- Terraform. (2018). Write, plan, and create infrastructure as code. Retrieved from <https://www.terraform.io/>.
- The getKanban Board Game. (2018). The getKanban Board Game. Retrieved from <https://getkanban.com/>.
- Trello. (2018). Trello lets you work more collaboratively and get more done. Retrieved from <https://trello.com/>.
- Vagrant. (2018). Development environments made easy. Retrieved from <https://www.vagrantup.com/>.

# Agile Approaches for Teaching and Learning Software Architecture Design Processes and Methods



Muhammad Arief Chauhan, Christian W. Probst and Muhammad Ali Babar

**Abstract** Software architecture plays a vital role in the analysis, design, evaluation and evolution of large-scale projects. Successful adoption of an agile methodology in large-scale projects requires not only tailoring of the software architecture analysis, design and evaluation methods but also a fundamental understanding of these methods. In this chapter, we provide agile teaching and learning approaches for software architecture analysis, design and evaluation. In particular, we focus on agile teams in architecturally significant (quality) requirements analysis and change management for collocated and distributed agile projects, iterative and continuous architecture design delivery using story boards and collaboration platforms, and using software reference architectures to monitor and control the design and evolution of a software architecture. The methods presented in this chapter are based upon the following research methods. We have explored the literature to identify key characteristics of agile software architecture processes and roles of agile teams in software architecture. We have presented agile teaching and learning approaches with reference to the case studies conducted in classes over 2 years of software architecture courses. We have specifically focused on designing course activities that can support lean education and collaboration among the students and course instructors. We foresee that the presented approaches can be used by academics to teach software architecture design methods and processes in particular, and software engineering techniques in general. Practitioners can also take advantage of the proposed approaches to continuously

---

M. A. Chauhan (✉)  
Netcompany A/S, Copenhagen, Denmark  
e-mail: [auch@netcompany.com](mailto:auch@netcompany.com)

C. W. Probst  
Unitec Institute of Technology, Auckland, New Zealand  
e-mail: [cprobst@unitec.ac.nz](mailto:cprobst@unitec.ac.nz)

M. A. Babar  
The University of Adelaide, Adelaide, Australia  
e-mail: [al.babar@adelaide.edu.au](mailto:al.babar@adelaide.edu.au)

educate their staff when applying agile methods for architecture design and evolution of complex software systems.

**Keywords** Agile learning · Software architecture · Software engineering  
Architecturally significant requirements (ARSs) · Architecture design  
Architecture evaluation · Architecture evolution  
Software reference architecture (SRA) · Internet of Things (IoT)

## 1 Introduction

Software architecture (Gorton, 2006) and agile software development (Hoda, Noble, & Marshall, 2013) have received ample attention from academics and practitioners. However, teaching software architecture in an agile context and learning to apply software architecture analysis, design and development methods for agile software projects remains a challenging undertaking. Adopting agile methods for teaching and learning software architecture requires teamwork, substantial coaching (Babar, Brown, & Mistrik, 2013) and tailoring of the software architecture analysis, design and evaluation methods. To equip the students with comprehensive understanding of the agile methods, it is important to enable them not only to learn in an agile environment but also to provide ample opportunities for the students to apply and practice the agile methods.

Agile software education and learning brings unique challenges with respect to achieving the learning objectives, selection and delivery of the course contents, and participation of the students in agile teaching and learning activities. Agile education objectives have to be in-line with agile software development (Sievi-Korte, Systä, & Hjelsvold, 2015; Angelov & de Beer, 2017) including but not limited to involvement of peers in learning activities, regular meetings, short learning cycles, continuous feedback and measurable artefacts at the end of each learning sprint. Introduction of electronic learning and collaboration platforms such as Moodle<sup>1</sup> and GitHub<sup>2</sup> can support collaboration among the students and course instructors.

Like other aspects of software engineering methods, software architecture practices in agile environment including agile education requires specific considerations. Software architecture describes not only different elements of a software system (such as classes, components and services) and relations among the elements but also encompasses the process and methods that are used for analysis, design, evaluation and evolution of a software architecture (Gorton, 2006). Traditionally, software architecture design processes have been categorized as activities that are carried out in the early phases of the software development life cycle. In the early days of agile software development, software architecture in agile projects was not explicitly focused because the agile process was being used during development of small-scale software systems only (Abrahamsson, Babar, & Kruchten, 2010). However, software architecture has become an important element with adoption of the agile

---

<sup>1</sup><https://moodle.org/>.

<sup>2</sup><https://github.com/>.

development methodology in complex projects, for which software development is carried out over a longer period. Software architecture can help to keep the focus on architecturally significant quality requirements can facilitate internal and external coordination, and can keep software development in line with core architecture design decisions (Abrahamsson et al., 2010).

A number of activities and design practices associated with software architecture need to be tailored and enhanced to educate practitioners and students for smooth adoption of software architecting processes in agile projects. Software architecture design and analysis activity encompasses architecturally significant requirement's analysis, identification of a system's elements (in terms of components, services, classes, persistence entities, etc.) and relations among the elements, software architecture evaluation and careful monitoring of an architecture's evolution to avoid software architecture erosion (Chauhan, Babar, & Probst, 2016b; Gorton, 2006). The high-level software architecture design focuses on architecturally significant quality (or non-functional) requirements, whereas detailed design focuses on quality as well as functional requirements. In non-agile software development approaches, high-level architecture analysis and design of the complete system is performed in early phases of a software development life cycle. However, as only features in the current sprint are addressed in agile projects, an overview of the overall architecture can get out of focus. Moreover, the absence of incremental analysis and impact analysis of the new features, which are added in each sprint can negatively impact overall quality of a system in general and architecture quality in particular.

Raising awareness of architecture decisions, as these are made during an agile project, is not trivial. In agile projects, rather than having an upfront detailed architecture design, architecture evolves as a project progresses. As a consequence, architecture design decisions are delayed until corresponding modules are developed. Communication among the team members in the form of face-to-face kick off meetings and daily stand-up meetings is a key characteristic of agile teams (Sievi-Korte et al., 2015). However, such meetings often encompass only a quick overview of the tasks and activities of the team members, and do not include details of architecture design decisions and choices made during day-to-day activities. Agile tools such as storyboards and project back-logs only focus on functional requirements. Architecture often remains invisible in agile projects, which can result in low-quality software in medium and large-scale projects. Hence, it is important to not only teach agile architecting methods to the students but also to teach the methods in an agile learning environment so that the students can practice agile architecture design and analysis methods in an environment that enables agile and lean learning.

To adequately address the afore-mentioned challenges, there is a need to have a specialized approach for teaching and learning software architecture analysis and design methods for agile projects. In particular, we focus on the following objectives in this chapter:

- We discuss the importance of agile and lean education with reference to software engineering in general and software architecture in particular. We discuss different dimensions of agile learning and teaching for software architecture-centric activities, which can facilitate analysis of the new courses and redesign of the existing



courses aiming at teaching agile software architecture analysis and design for students and practitioners.

- We analyse the impact of agility, teams distribution and complexity of the projects on architecture quality, which is elaborated in terms of Architecturally Significant Requirements (ASRs), e.g. security, scalability and reliability. The impact analysis can help to select methods corresponding to the activities that are related to key learning objectives of a course.
- We present agile teaching and learning methods for software architecture analysis, design and evaluation. We elaborate application of the presented methods with reference to case studies that have been conducted in graduate courses with a primary focus on software architecture analysis, design and development. We also discuss in a broader context how the presented methods can be used for teaching and learning software engineering topics. We discuss how the presented methods and case studies of their application can provide a practical guide for adoption of the agile methods for tertiary education.

This chapter is organized as follows. Section 2 provides an insight to the key dimensions of agile learning. Section 3 builds foundation for the presented agile learning approach by describing an analysis of key dimensions of software architecture with reference to agility and how agility impacts activities associated with software architecture analysis, design and evaluation. Section 4 elaborates methodology for agile software architecture education and learning. Section 5 describes application of the proposed agile education and lean learning methodology with reference to a course on software architecture. Section 6 summarizes our experiences of the case studies and discusses applicability of the proposed methods for software engineering education in broader context. Section 7 discusses related work and Sect. 8 concludes this chapter.

## 2 Key Dimensions of Agile Learning

Different dimensions of agile and lean methods have specific relevance to software engineering education in general and software architecture education in particular. In this section, we provide an overview of the key elements and roles in agile processes, correspondence of the roles to agile education and insight on the structure of a software engineering focused course for incorporating agile and lean learning.

### 2.1 Key Elements and Roles in an Agile Process

Agility is regarded as ability of an organization, team or a person to adapt and respond to changes in its operating environment to make profit (Abrahamsson et al., 2010). The profit can be a direct reward such as monetary benefit or an indirect reward such as an increase in organizational productivity. Self-organization is one of the key

characteristics of agile teams. Self-organizing teams are formed spontaneously to work on a task, are not part of the formal organizational structure, and have a shared purpose (Hoda et al., 2013). Moreover, while keeping the focus on the team goals, individual team members make decisions for their own tasks (Hoda et al., 2013). Even though teams are self-organizing, the role of mentor in agile teams cannot be underestimated. An agile team member can perform one or more of the following roles: mentor, coordinator, translator, promoter and terminator (Hoda et al., 2013). The *mentor* guides and supports the development and team members during initial set-up and encourages self-organization practices during execution of the tasks. The mentor provides not only guidance and support for continuous adherence to the agile principles but also helps to remove misconceptions of agile activities (Hoda et al., 2013). The *co-ordinator* manages and coordinates customers expectations with the agile team members, whereas the *translator* translates the business language used by the customers to technical terms used by the team members. The *promoter* promotes the agile methodology with the customers and the *terminator* facilitates the movement of unsuitable team members, who do not conform to the agile development practices, away from an agile team.

In agile teams, one's knowledge is not limited to one's own self. Members of an agile team use and take advantage of the knowledge of other team members as well (Nevo & Chengalur-Smith, 2011). As a result, communication and knowledge sharing among the agile team members play a vital role in agile processes. The capability of a communication medium and the process that is used for communication are also critical to smoothly carry out activities associated with an agile project (Nevo & Chengalur-Smith, 2011). Shared patterns of communication are evolved, and coordinated behaviours are established among the agile team members as they progress and work together on the shared tasks. Communication media can also facilitate awareness of the activities among the team members. Another attribute of the communication medium is to provide a sense of social presence among the team members with as little effort by the team members as possible. Convergence of the communication media facilitates conflict resolution and problem-solving (depending upon the convergence ability of a communication medium). Hence, the stronger the communication attained by a team's members, the better they perform in agile settings.

## ***2.2 Correspondence of Agile Education to Key Agile Elements and Roles***

The key elements and roles of agile (as discussed in Sect. 2.1) are also relevant to agile education and lean learning. To enable students to work in agile settings, learning activities are to be carried out in groups of students, which can be analogous to teams in an agile process. Unlike traditional classroom teaching, agile teaching encompasses a number of cohesive and loosely coupled tasks, which signifies the

importance of self-organization among the agile student teams. The teaching staff members can perform the roles of mentor, coordinator, translator and promoter by enabling the students to carry out learning activities in agile settings. As learning activities following agile methods are carried out in groups, communication among the team members and teaching staff, as well as use of communication media for awareness, conflict resolution and problem-solving is important. Some important aspects of agile software engineering education is pair programming, face-to-face meetings or meetings using digital platforms, clearly defined roles and well-defined distribution of work tasks (Sievi-Korte et al., 2015).

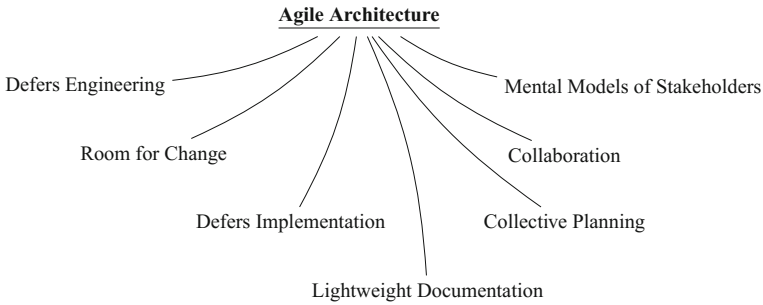
### ***2.3 Structure of a Software Engineering Course for Agile and Lean Learning***

For enabling the students to learn agile software engineering and agile software architecting, the course should provide a platform where the students can engage in activities characterizing agile software development. These activities include inter-group and intra-group collaboration, face-to-face and digital communication, clarification on the semantics of the course contents, incremental feedback to the students on course assignments and projects, and clear description of inter-group and intra-group activities. Angelov and de Beer (2017) have emphasized the need to explain the following points to the students in an agile learning environment:

- project context and objectives,
- roles that are performed by the stakeholders in real-life scenarios,
- nature and type of the documentation to be used in the projects and
- explicit explanation of the specific methods that are to be used by the students to carry out learning activities in a course.

## **3 Agility, Software Architecture and Lean Learning**

Architectural activities in a project vary widely depending upon the project domain, organizational structure, project size, business model and rate of change in a system under development (Abrahamsson et al., 2010). Activities related to software architecture can be classified generally into four main categories: (i) architecture analysis, (ii) architecture design, (iii) architecture evaluation and (iv) architecture evolution (Gorton, 2006). The architecture of a software intensive system also reflects organizational structure, application domain and mental models of the stakeholders (Coplien & Bjørnvig, 2011). In non-agile and incremental software development models, initial phases of the projects have extensive focus on software architecture including (i) high-level architecture design in terms or architecture styles and patterns and (ii) detailed architecture design in terms of design patterns, frameworks and



**Fig. 1** Agile architecture elements (Coplien & Bjørnvig, 2011)

programming languages, data structures and persistence approaches. Many of the architecture decisions can be hard to undo at later stages without a major re-factoring. Software architects perform the role of a liaison to bridge the gap between business and technical stakeholders, support high-level and detailed architecture design activities, evaluate software architecture designs and monitor architecture evolution to control architecture erosion.

Contrary to non-agile software design, agile software design is characterized by postponing the architecture decisions as late as possible and to provide flexibility to make changes in the design at later stages. Organizational as well as software architectural analysis and design activities for agile development differ from non-agile development in terms of (Coplien & Bjørnvig, 2011):

- when the architecture decisions are made,
- what architecture decisions are documented,
- how an agile project is planned and managed and
- how people are organized in an agile project in terms of roles, responsibilities and task assignments.

Each of the afore-mentioned questions is handled differently in agile architecting. Engineering activities are deferred until the time the related requirements are to be implemented in an agile project. A classic software architecture design tries to limit the changes, whereas an agile software architecture provides room for changes in the architecture while detailed system requirements are analysed and implemented. Software architecture documentation is also limited to key design decisions in agile projects (Abrahamsson et al., 2010). Team members are assigned clearly defined roles during different phases of an agile project, which might be different from that of the organizational hierarchy. Collective planning and cooperation is ensured by the managers to keep a project's activities on track (Coplien & Bjørnvig, 2011). Instead of focusing on technical details such as coupling and cohesion of the detailed architecture elements (e.g. components and classes), an agile architecture focuses more on end users' mental models, which can inspire implementation of the project at later stages (Coplien & Bjørnvig, 2011). The characteristics of agile software architecture process is presented in Fig. 1.

The lean and light weight nature of the agile architecture design practices impose a number of challenges for teaching agile architecture design processes. Educating students about incremental design of comprehensive systems architecture in complex domains such as providing software design and implementation Tools as a Service (TaaS) following pay per use model (Chauhan, Babar, & Sheng, 2015), require interactive learning methods requiring active participation of the students. Especially, when such systems are used as part of the collaborative workspaces and have high degree of dependency upon other systems (Chauhan, Babar, & Sheng, 2017).

### ***3.1 Activities of the Software Architects in Agile Projects***

Software architects in agile projects need to focus on traditional as well as agile specific activities. Traditional activities include describing the elements of a software system and relationship between the elements, and establishing the processes and methods that are used for software architecture analysis, design, evaluation and evolution (Chauhan et al., 2017; Gorton, 2006). Agile specific activities include (Coplien & Bjørnvig, 2011):

- (a) Focusing on essence of the system rather than functionality.
- (b) Defining components and subsystems according to their rate of change as well as functionality.
- (c) Identifying components and subsystems so that each of these can be managed as autonomously as possible.
- (d) Focusing on locality of the changes.
- (e) Prioritizing human considerations while driving the partitioning for software engineering concerns.

The agile specific activities are particularly important to structure the architecture and encompass the activities that are suitable for agile development. Defining components and subsystem based upon the expected rate of change and functionality helps to achieve point b, i.e. to separate parts of the system requiring rapid development and changes from that of more stable areas of the system. Defining modularity based upon the rate of change also help to achieve point c, i.e. autonomous development on each module without having significant impact and interdependency upon other modules. Keeping the domain knowledge of a specific part of the system in the same geographic locations and within the same architecture unit during design and development of a system positively contributes to achieve cohesion. Partitioning of modules following the domain knowledge also achieves cohesion. If the architects do not have a comprehensive understanding of the domain while architecture is being designed, they can use end user cognitive models of the domain to derive a system's modularity (Coplien & Bjørnvig, 2011). Using product lines development approaches for domain partitioning can also facilitate reuse of the system components and domain knowledge to a certain extent (Coplien & Bjørnvig, 2011), i.e. to achieve points d and e.

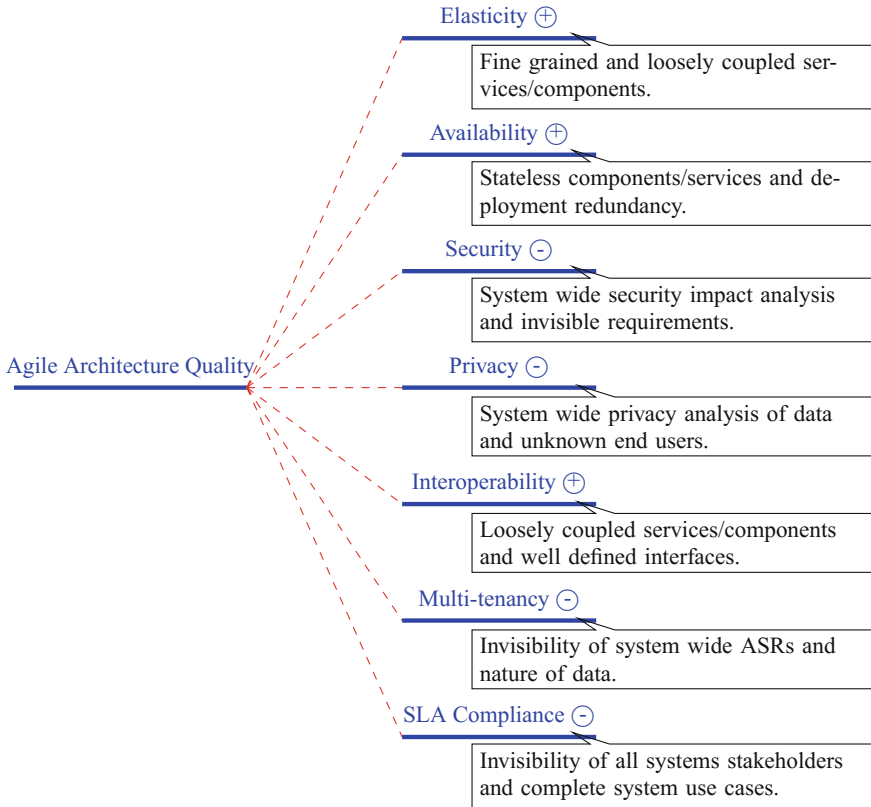


Fig. 2 Impact of agility on software architecture quality

### 3.2 Impact of Agility on Architecture Quality

Based on the principles of agile software development, agile architecture evolves as different parts of the system are designed and implemented. As software architecture primarily focuses on architecture quality attributes (also referred as architecturally significant or non-functional requirements), some quality requirements are positively impacted by architecture agility, whereas other requirements are negatively impacted by architecture agility. Figure 2 shows a pictorial representation of the relation between architecture quality attributes (Chauhan, Babar, & Benatallah, 2016a) and agility. A description in the box corresponding to each quality attribute explains what influences agility (positively ⊕ or negatively ⊖) for the respective quality attribute. An agile learning approach for software architecture education should enable the students to learn how to focus on desired architecture quality attributes.

### ***3.3 Impact of Distribution of Agile Teams on Software Architecture***

Agile teams focus on coordination, collaboration and communication, also referred as three ‘C’s of agile (Dingsøy, Dybå, & Moe, 2010). The agile teams can be distributed or co-located. Agile teams need to make architecture decisions within teams and communicate the decisions with other teams. Hence, the three ‘C’s have a key place in agile architecture development in different phases from elicitation of the architecturally significant requirements to the detailed architecture design. Agile teams need to communicate architecture design decisions that are made locally to other distributed and co-located teams, and need to collaborate with each other to resolve conflicting architecture design decisions. Agile teams’ distribution signifies the importance of having architecture control mechanisms as well. The control mechanisms make sure that architecture design choices that are made by each individual team are in line with system-wide software architecture design and quality objectives. Hence, lean software architecture education method should focus on three C’s of agile.

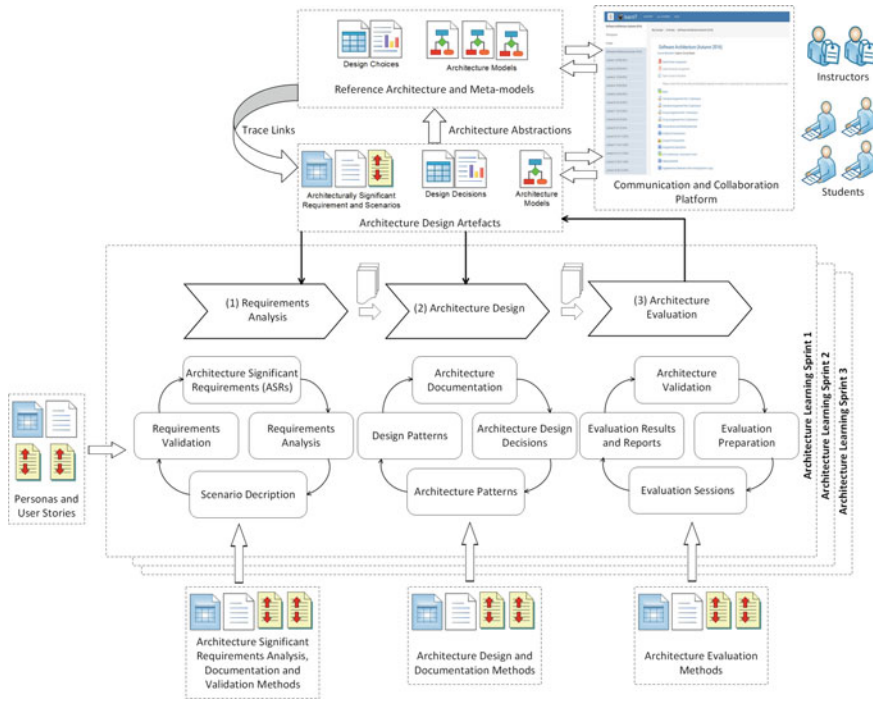
### ***3.4 Impact of Complexity and Domain of the Projects on Agile Processes***

The complexity of the domain of a specific project also impacts the process that is adopted for agile software architecture analysis, design and evaluation. For complex domains, core design decisions and design artefacts, which are central to the system should be controlled and governed by a core team of architects with domain expertise.

## **4 Methodology for Agile and Lean Software Architecture Education**

A comprehensive methodology for agile and lean learning should encompass all important topics for software architecture and organize the course to incorporate all the key elements of an agile process. The methodology should provide an opportunity to the students so that they can perform software architecture analysis, design and evaluation activities following agile and lean processes.

Incorporating agility in software architecture education requires incremental and iterative delivery of the course contents, and quick feedback on the student’s deliverables. Heavyweight software architecture design and evaluation methods need to be tailored to match specific needs of agility in software architecture. In this section, we describe an agile education strategy for software architecture intensive courses. We also describe strategies for software architecture design and evaluation activities,



**Fig. 3** Agile teaching and learning process for a course on software architecture

which are tailored to maximize engagement of the students in the learning activities and bring the learning experience closer to real-life agile projects.

An aggregated view of the agile architecture education and learning process is presented in Fig. 3. The centre of the figure shows learning activities that can be included in each sprint of the agile software architecture learning process. The students can be taught requirements elicitation, architecture design, and evaluation activities corresponding to the selected set of user stories in each sprint. Related literature (depicted in the figure as an input to the course activities) can be taught to perform the activities in accordance with the schedule of a course. The teaching and learning activities can be organized based upon the complexity of the course topics. Each sprint can have more than one iteration, and each proceeding iteration can have more complex course contents along with exercises as compared to previous iterations.

Electronic communication and collaboration platforms, which can be used by the course instructors and students, are represented at top right of the figure. All the exercises and assignment descriptions, deliverables submitted by the students, feedback on the deliverables by the teaching staff and preparation notes along with results of the evaluation sessions can be managed using the communication and collaboration platform.



In the following subsections, we first describe the key elements of a software architecture course for agile and lean learning. We then provide details of the key elements of the agile and lean learning methodology specific to software architecture education.

#### ***4.1 Key Elements of the Agile Software Architecture Course and Students' Activities***

Designing software architecture involves: (i) identifying requirements that can have an impact on a software architecture, (ii) detailed design and presentation of the architecture to represent structure and behaviour of a system and (iii) evaluating the candidate architecture solution to make sure that it can comply with the desired functional and quality requirements (Gorton, 2006). For long-term maintainability and evolution of the software, software architecture evolution needs to be monitored and controlled to reduce the risk of architecture erosion (Chauhan et al., 2016b).

***Architecturally Significant Requirements Elicitation, Documentation and Management:*** Elicitation, documentation and management of Architecturally Significant Requirements (ASRs) is a key activity of the architecture design process. It involves identification of architecture savvy personae (Cleland-Huang, 2013) and capturing their user stories for functional and quality aspects of the system. Each persona represents a specific stakeholder of a system. It can be a domain expert, a software developer or an end user. Each persona specifies what is acceptable and what is not acceptable for them from the system under design. The system's expected behaviour is specified in terms of user stories. The user stories can capture both functional and non-functional (also referred as ASRs) requirements. However, for software architecture perspective, only ASRs are important.

The ASRs specified by the users are described in terms of three attributes: (i) stimulus, (ii) response and (iii) response measure (Clements et al., 2002). The process of refining the requirements helps to identify incomplete requirements. For example, if an ASR is associated with availability and it does not provide any metric to measure satisfiability of the requirement (e.g.  $24 \times 7$ ), then the requirement is incomplete. The refinement process also helps to identify complementary or conflicting requirements (Chauhan et al., 2016b). For example, if availability ASR states that the system should be available  $24 \times 7$  and maintainability, ASR states that the system should be down while installing new upgrades, then there is a conflict.

***Software Architecture Design:*** The artefacts produced as a result of software architecture design activity are used as a baseline for software development. The design artefacts can correspond to any of the logical, process, development or deployment view (Kruchten, 1995). These views include different types of design diagrams, e.g. class diagrams are a part of logical view, sequence and collaboration diagrams are a part of process view, component diagrams are a part of development view, and deployment diagrams are a part of physical view (Kruchten, 1995). The primary

activity of the design phase is to incorporate a number of architecture and design patterns to achieve architecture quality, i.e. to satisfy the ASRs.

**Software Architecture Evaluation:** The purpose of architecture evaluation activity is to make sure that the architecture is compliant to the desired quality, i.e. all the important ASRs are properly transformed into the architecture design documents. A number of software architecture evaluation methods such as Architecture Trade-Off Analysis Method (ATAM) (Kazman et al., 1998), Software Architecture Analysis Method (SAAM) (Kazman, Bass, Webb, & Abowd, 1994) and Quality-Driven Architecture Design and quality Analysis (QADA) (Matinlassi, Niemelä, & Dobrica, 2002) can be adopted based on the requirements of a specific project.

**Monitoring and Controlling Software Architecture Evolution:** Control over the process of software architecture evolution is important to protect the software architecture from erosion and deviation from the long-term architecture quality objectives. Hence, it is an important course topic. Though it is challenging to measure the impact of change on software architecture, using software reference architecture to capture a blueprint of the architecture and adaptation of semi-formal approaches such as attack-defence trees can help to monitor architecture evolution (Chauhan et al., 2016b).

Different course topics and key elements of each topic are summarized in Table 1.

## ***4.2 Iterative Delivery of the Course Contents Combined with Short Hands-On Exercises***

In order to incorporate agility in software architecture education, rather than following a waterfall delivery approach, the course topics should be delivered iteratively. That requires splitting up course contents on the basis of the difficulty level of each topic (e.g. ranging from basic to advanced) and grouping the course contents from the different topics based upon the difficulty level. For example, the process of requirements elicitation covers architecturally significant requirements identification and documentation (in terms of quality to be achieved, stimulus of the requirements, response of the system when the requirement is implemented in the system and metrics for response measure) (Gorton, 2006).

For agile learning, in the first phase of the architecture design sprint, only quality attributes corresponding to the most important quality requirements can be focused on, e.g. scalability or elasticity. Next, architecture scenarios in terms of stimulus, response and response measures can be taught. In the second phase, the quality attribute can be taught and combined with the high-level architecture design tactics and architecture patterns. For example, scalability and elasticity can be achieved using multi-tier architecture and stateless system services. Multi-tier architecture and stateless services facilitate adjustment in the runtime configuration of the system as the system needs to be scaled up or down. In next phase, stimulus, response and response measures can be taught in combination with detailed design tactics

**Table 1** Software architecture course contents and elements

Course topic	Element	Description
Personae	Profile	Details on a persona including picture, name, designation, education, professional background and role in the system under development
	User stories (what is expected)	What functional and non-functional (quality) requirements a persona expects to be implemented in the system
	Anti stories (what is not expected)	System behaviour that is not expected by a persona
	Quality attributes of interest	A list of quality attributes that a persona is interested to have in the system
Architecturally Significant Requirements (ASRs)	Quality	A specific quality or a non-functional requirement, e.g. scalability, reliability, security, privacy, etc.
	Scenario	Description of the quality requirement in terms of stimulus (what triggers a quality requirement), response (what should be the system's behaviour satisfying the quality requirement) and response measure (metric to measure the quality response)
Architecture design	Design views	Multiple views of software architecture design including process view, logical view, development view and physical view
	Diagrams in each view	Diagrams corresponding to different views including sequence and activity diagrams for process view; service, class and state diagrams for logical view; component and package diagrams for development view; and deployment diagram for physical view
Evaluation	Software architecture evaluation methods	Evaluating software architecture using methods such as ATAM, SAAM and QADA
Architecture evolution	Controlling architecture evolution	Methods and techniques to monitor and control software architecture evolution

for scalability and elasticity such as cache, facade, redundant system deployment, data synchronization methods, and distributed computing (Buschmann, Henney, & Schmidt, 2007). The whole process can be repeated in the second and third sprints for user stories and quality requirements that are more difficult to achieve. A course can have multiple sprints depending upon the duration of the course and education background of the students.

### ***4.3 Using Digital Platform(s) for Communication, Collaboration and Feedback***

Communication, collaboration and self-organization are key practices of agile teams (Campanelli & Parreiras, 2015). When multiple agile teams are working on related or dependent tasks, they frequently need to communicate and collaborate with each other. The same is true for student teams and teaching staff for agile software architecture education. Use of digital platforms for communication, collaboration and feedback on assignments and project deliverables can support frequent interaction and rapid exchange of the artefacts among the student team members, and between student teams and teaching staff. Digital platforms can also facilitate teaching staff to share learning material related to a specific phase of the agile education project and can provide a single point of access for all the relevant course information.

### ***4.4 Incremental Deliverables and Rapid Feedback***

A key strategy for incorporating agility in software architecture education is to have iterative deliverables of the students' assignments and projects, following by a rapid feedback on the deliverables. A key step in this regard is to devise a strategy for assignment and project delivery so that output of the preceding phase can be used as an input to the proceeding phase.

For example, Architecturally Significant Requirements (ASRs) (Chauhan et al., 2016a) elicitation and management process can be divided into the following phases. In the first phase, the students can be asked to capture the user stories (Gorton, 2006), i.e. to identify what can be potential users of a given software system and what can be their expectations (in terms of features) from the system. In the second phase, the students can be asked to identify ASRs (quality requirements or non-functional requirements) from the user stories. In the third phase, the students can be asked to capture the details of the ASRs in terms of stimulus (source of the requirement), response (outcome if the requirement is incorporated in the system) and response measure (how to verify if the requirement is implemented correctly) (Clements et al., 2002).

Software architecture can be incrementally designed corresponding to the requirements' elicitation and management phases. After the first and the second phase of requirements elicitation, high-level architecture decisions can be made. After the third phase of requirements management, detailed architecture design decisions can be made such as using model-view-control pattern (Buschmann et al., 2007) to increase modularity and modifiability, or using redundant deployments of the system's services (Buschmann et al., 2007) to increase availability.

## 4.5 *Learning Software Architecture Design*

Agile software architecture design encompasses the incremental design of different aspects and views of a software architecture (Kruchten, 1995), which gradually contributes to a complete system architecture. Agile and lean architecture learning can be achieved by combining small class exercises on software architecture design and by complementing the exercises with dedicated architecture design evenings.

The first step for agile architecture design is to use the personae-based approach to capture user stories (as discussed in Sect. 4.1) which can later be used to refined ASRs and architecture scenarios. The personae can be an important input for prioritizing ASRs for different sprints of the agile design process. The personae can be helpful to resolve conflicting ASRs because these can identify the sources of the ASRs.

The following architecture design activities can include transforming ASRs into concrete architecture scenarios, identifying relevant architecture patterns (Buschmann et al., 2007) to satisfy high-level ASRs, selecting design patterns (Shalloway & Trott, 2004) to material detailed architecture design, and selecting appropriate views of 4 + 1 architecture view model (Kruchten, 1995) that best suit the domain of a system being designed. In the first sprint, primary ASRs, appropriate system view and high-level architecture patterns can be decided. In the second sprint, while the detailed design is elaborated in terms of design patterns and concrete implementation strategies based upon the output from the first sprint, ASRs from the remaining pool of the requirements can be selected.

A key practice for the agile architecture design is to maintain a backlog of all the important decisions made and traceability between ASRs and architecture design decisions (architecture patterns, design patterns and concrete scenarios corresponding to the architecture significant requirements). The scrum meetings can be organized at the beginning of every exercise session, where group members quickly review and analyse ASRs and corresponding architecture scenarios, architecture patterns and design patterns that other members of the group have worked on. The project backlog can be the list of tasks that the students are expected to perform throughout the course. The project backlogs should be shared between a group and teaching staff. In the beginning of the course, the project description and tasks list should be shared with the students.

One of the key characteristics of the agile project is continuous delivery of a product under development. For a semester-based course on software architecture, the course can be divided into two sprints of 5 weeks each. The first sprint can begin after 2 weeks of the first lecture. After each sprint, the students submit a formal project report. Within the sprint, the teaching staff can have semi-formal meetings with each student team to track the progress. Inter-group and intra-group correspondence can be carried out through a digital platforms, e.g. by using Moodle.<sup>3</sup>

---

<sup>3</sup><https://moodle.org/>.

## 4.6 *Learning Agile Software Architecture Evaluation*

Software architecture evaluation is a key activity of the software architecting process. Hence, it should also be a part of the agile learning activities in a software architecture course. To make architecture evaluation part of the agile learning process, 3 weeks period can be scheduled between the sprints (considering that a course has two sprints). The output of the first sprint can be used as an input for the evaluation activities and output of the evaluation activities can be used in sprint two for refinements in the architecture design along with incorporation of the new requirements. The number of sprints in a course can be increased depending upon the course duration and contents.

Architecture evaluation sessions should be organized in such a way that members of each group participate in at least two evaluation sessions. In the first evaluation session, the group members can participate to present the key areas of their architecture design and get their architecture design evaluated. In the second session, they can evaluate the architecture of another group and provide feedback on the designed architecture. The architecture of the system that is to be evaluated, should be shared with the evaluation team members in advance (e.g. 1 week before the evaluation session) so that they can have a good understanding of the system design. For example, if group B is to evaluate group A's architecture, group A's architecture should be shared with group B. Evaluation sessions can be organized by following guidelines for any of the architecture evaluation methods [such as ATAM (Kazman et al., 1998), SAAM (Kazman et al., 1994) and QADA (Matinlassi et al., 2002)], depending upon which method suits the project. Each group has to prepare two reports. Before the evaluation session, the group that is evaluating the architecture (group B) has to present a short list of architecture concerns (ASRs, patterns, architecture diagrams, etc.) that they want to focus on during the evaluation session. After the evaluation session, each group that has evaluated an architecture (group B) has to prepare an evaluation report which can be used as an input for the second sprint.

## 4.7 *Learning Agile Software Architecture Evolution*

Controlling architecture evolution when multiple teams work on designing related and dependent subsystems in large-scale software systems is not trivial. In agile projects, architecture evolves in each sprint. Therefore, it is critical to monitor and control the architecture evolution to keep it on track and avoid architecture erosion. Reference Architectures (RAs) (Chauhan et al., 2016b) can be a useful tool to monitor and control architecture evolution. Depending upon the maturity of the domain, a RA can be adopted to control evolution in two different ways.

- For mature domains (where one or more comprehensive RAs already exist), a RA that is more closely related to the project scope and domain, can be selected. The selected RA can be used as a starting point to guide the design of the software

system to be developed. For each sprint, trace links of the detailed architecture artefacts (including personae, ASRs, architecture design decisions, architecture patterns, design patterns and different views of the architecture) should be established with the RA and this information should be shared among all the student teams working on the system (in project settings when multiple teams are working on different parts of the same system) and teaching staff.

- For domains in which comprehensive RAs are not available, an abstraction of architecture can be derived from the detailed design after each sprint. This abstraction can contain trace links to the detailed artefacts. This step should be repeated after each sprint.

As a project progresses, the RA can server as a single point of access for all architecture artefacts designed in each sprint. As the new artefacts are added, trace links to the new artefacts are added and previous links are updated if needed. Relations between different types of the artefacts and their impact upon each other can be analysed by the trace links. The RA along with the trace links to the detailed artefacts can be used to identify conflicting and complementing architecture decisions. For example, ASRs can be evaluated by using a probabilistic analysis method to analyse the impact of the ASRs on each other in terms of whether the requirements are dependent on each other, complement each other, or contradict each other (Chauhan & Probst, 2017). The continuous impact analysis allows monitoring of the evolution of the architecture as it matures and protects it from erosion even if the complete architecture is not designed upfront (in one go).

## 5 Case Studies on Application of the Proposed Methodology

The agile and lean software architecture methodology presented in Sect. 4 was used in a Software Architecture course for a Master's Degree in Software Development Technologies at IT University of Copenhagen (ITU) Denmark<sup>4</sup> in the fall 2015 and 2016 semesters. There were 35 students on average in the class in both years, and 3 course instructors. One of the course instructors was course manager and main lecturer. The other two instructors were responsible for delivery of lectures on specific topics and execution of the exercises. In this section, we describe details on application of the presented approach in the course.

---

<sup>4</sup>[www.itu.dk](http://www.itu.dk).

## 5.1 Course Structure and Distribution of the Roles

The courses consist of 14 weeks of teaching with 1 week of teaching break after the 17th week. In both years, the course was divided into 2 sprints. The first 2 weeks of the course were dedicated to an introduction to basic topics on software architecture, the Unified Modelling Language (UML)<sup>5</sup> and formation of the students working groups. Each group had at least three members. The groups were free to choose the roles within the group (e.g. scrum master and architect team members). The first sprint was organized from week three till week seven. Weeks within the teaching break and the eighth week of teaching were allocated for preparation of the software architecture evaluation sessions (in which each student group evaluated architecture of another group). In addition to the evaluation sessions, the teaching team also evaluated the architecture of each group. The second sprint was organized from week 9 till week 13. In the 14th week, the teaching team evaluated the final architecture report of each group and shared evaluation findings with the respective groups.

The course in both sessions (2015 and 2016) covered the following topics (as described in Sect. 4.1). The students are asked to use UML and Service-oriented architecture Modelling Language (SoaML)<sup>6</sup> to design the architecture diagrams. The students were given the tasks to design the architecture of a part of the Internet of Things (IoT) for smart homes. Only the high-level requirements of the system were shared with the students. The students were asked to choose a part of the IoT smart home system. For example, electricity plus appliance management and control of entrance into the home were the areas chosen by majority of the groups. All members of the teaching staff were actively involved in reviewing the students' intermediate submissions and providing feedback on the submissions.

## 5.2 Digital Platform Structure

The digital course management platform LearnIT<sup>7</sup> was used in the course for sharing the course material, updates on daily activities, and collaboration among the students and course instructors. Figure 4 shows a screenshot of the course home page on LearnIT. The course page had areas for submission of the students' deliverables (individual and group assignment submission areas) as well as discussion forums (e.g. IoT Architecture Discussion Forum). The course page had sections for each lecture session where details on the learning and exercise materials were shared with the students by the course instructors. There was also a news forum on which teaching staff and the students could post important details on the course.

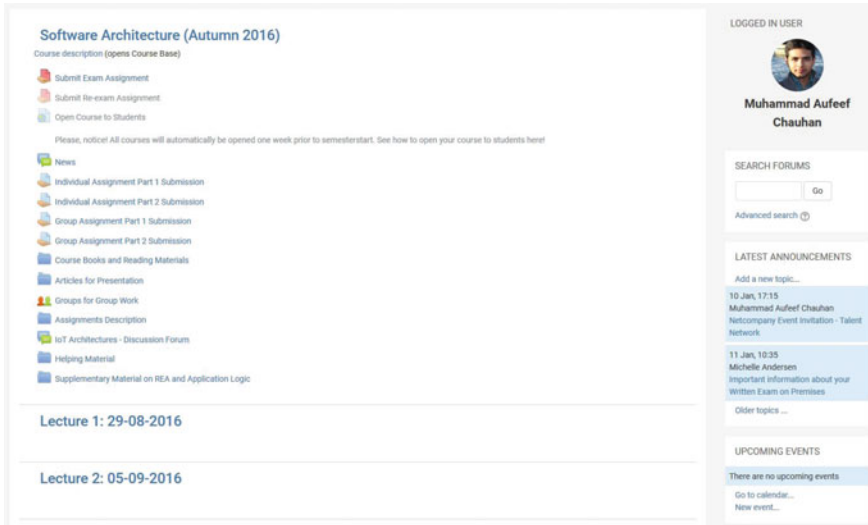
---

<sup>5</sup><http://www.uml.org/>.

<sup>6</sup><http://www.omg.org/spec/SoaML/About-SoaML/>.

<sup>7</sup><https://learnit.itu.dk>.





**Fig. 4** LearnIT—a digital content sharing and collaboration platform

### ***5.3 Weekly Architecture Analysis and Design Sessions Using Drawing Boards and CASE Tools***

There was an exercise session each week during both sprints. A dedicated area was allocated to each student group where they could work on the given tasks. The students used drawing boards and Computer Aided Software Engineering (CASE) tools to design architecture artefacts. The tasks carried out during each exercise session contributed to the detailed architecture design of the respective sprint. In the first sprint, the students were asked to focus on simple ASRs (such as modularity, scalability, etc.) along with functional requirements. In the second sprint, the students were asked to focus on more complex ASRs (such as security, reliability, etc.). In the beginning of the first sprint, the groups were asked to define personae for their respective part of the system. As an example, personae identified by one of the groups is shown in Fig. 5. In the beginning of the second sprint, the groups were asked to refine the personae for the additional quality requirements. However, during each sprint, the groups were also improving the personae and user stories following the feedback from the teaching staff. After identification of the personae and extraction of the ASRs from the personae, the groups designed the architecture using UML and suitable design plus architecture patterns in each sprint.

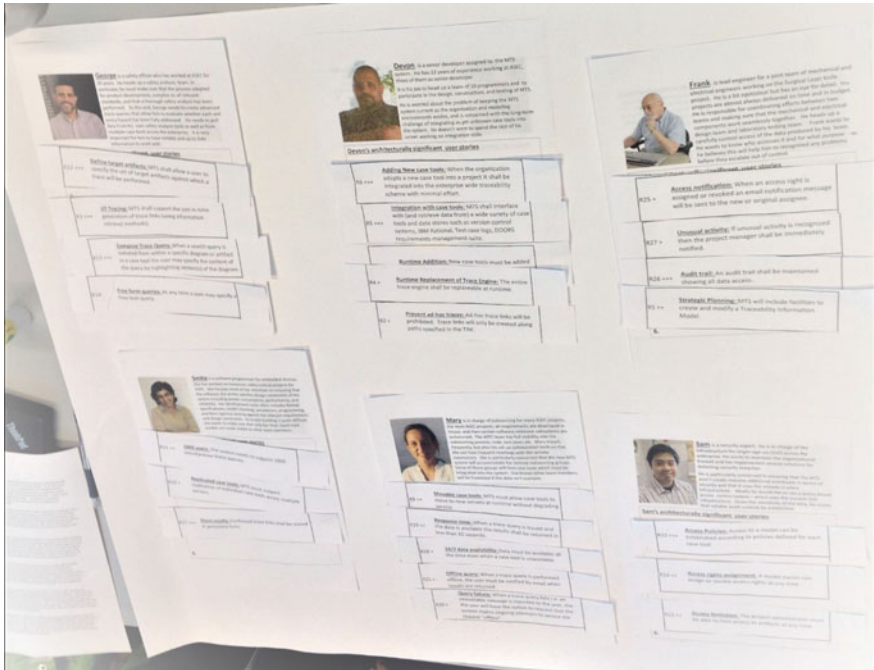


Fig. 5 Example architecture personae generated during weekly exercises

### 5.4 Deliverable and Feedback Cycles

The students were given a quick feedback on their deliverables at the end of each exercise session. The students were asked to deliver a short informal presentation (of 5–10 mins) to one of the teaching staff members, describing the architecture analysis and design activities they performed along with outcome of the activities. The teaching staff members provided feedback on the produced artefacts. The students were then asked to submit the corrected artefacts in the assignments and projects submission folder as part of the continuous delivery of the architecture design. At the end of each sprint, there was a formal feedback on the final submitted report of the sprint following a separate discussion with each group. The same process was repeated in the second sprint.

### 5.5 Architecture Design Sessions

In addition to the weekly exercise sessions, there was also a dedicated session on the design in the middle of each sprint for which the groups were asked to make modifications in the design following on the spot input from the teaching staff.

The students worked on the same part of the IoT subsystem during design session as part of their project. The students were asked to use drawing boards, case tools and flip charts so that they could engage in intra-group communication effectively. Design sessions consisted of only intra-group activities plus reviews and feedback from the teaching staff.

## 5.6 Architecture Evaluation Sessions

Architecture evaluation sessions were scheduled at the end of the first sprint and before the beginning of second sprint. The evaluation sessions were organized so that each group's architecture was evaluated by another group. For this purpose, the architecture design document produced as a result of the first sprint was shared with the group who was going to evaluate the architecture, 1 week prior to the evaluation session. The students were asked to prepare two reports for each evaluation session. (i) A short report (one to two pages) describing their initial analysis of the architecture prior to the session. This report was used as a guide for the students during the evaluation session. (ii) A second report (four to five pages) describing evaluation results of the architecture after the evaluation session. The students were asked to submit the first report before the evaluation session and second report 1 week after the evaluation session. Both of these reports were individual tasks and each student was asked to submit the reports as part of their individual assignments. However, the students were encouraged to collaborate with each other before, during and after the evaluation sessions.

The students were given a rough guideline on organization of the evaluation sessions. Each evaluation session was scheduled for one hour and thirty minutes. The organization of the evaluation sessions is presented in Table 2. The groups were

**Table 2** Organization of each evaluation session

Time (min)	Activity	Description
10	Introduction	Members of the architecture evaluation session introduced each other and their core expertise
30	Presentation of the architecture	The group whose architecture design was to be evaluated gave a presentation to the group who was evaluating their architecture
30	Questions and discussion	The group who was evaluating the architecture asked questions and presented their view on different parts of the architecture
20	Debriefing and notes	The groups summarized the findings of the session and took notes to be used in the evaluation reports

allowed to use any of the evaluation methods such as ATAM, SAAM, QADA or their own tailored evaluation approach. However, they were asked to provide the details on the process they followed in their evaluation report. The students were asked to incorporate feedback of the evaluation sessions along with feedback provided by the teaching staff in the second sprint of the architecture design. There was no dedicated evaluation session after the second sprint and the teaching staff evaluated the architecture deliverables submitted by the students.

### ***5.7 Using Architecture Meta-models and a Reference Architecture to Support Architecture Evolution***

A key activity to manage agile architecture evolution is to explore the availability of Reference Architecture (RA) and meta-models so that an appropriate strategy to handle traces and variability in the architecture can be adopted (as discussed in Sect. 4.7). Because of the availability of the RA for IoT, the students were given the reference architecture presented by Boussard et al. (2013). The students were asked to follow the high-level architecture description presented in the RA to design the architecture skeleton of a particular area of the smart home system they chose and establish its trace links to the reference architecture abstractions. As all the groups were using the same RA as a baseline, the students were asked to share their design choices for a particular area of smart homes system with reference to the RA on the discussion forum. The course manager and main lecturer also maintained an aggregated enhanced version of the RA for smart homes IoT, based upon abstractions from Boussard et al. RA (Bauer et al., 2013) and by using the students' architecture deliverables at the end of each sprint. The enhanced RA enabled the students to learn how their architecture fits into overall aggregated smart homes system domain.

## **6 Students Feedback and Discussion on Application of the Proposed Methodology on General Software Engineering Education**

Feedback from the students participating in the case studies (described in Sect. 5) reported a positive feedback of the proposed agile and lean software architecture design processes and methods. Table 3 shows average evaluation of the course material and teaching methodology by industrial students from ITUs official Software Architecture course evaluation of 2015 session. The evaluation was conducted via online questionnaire. Each of the students selected a value (between 1 and 6, where 1 is least positive) corresponding to a question. The questions along with average score corresponding to each question is shown in Table 3. The evaluation results show a positive feedback and all the students (in both years) who appeared in the final exam passed the course. Hence, it can be concluded that the adopted methods

**Table 3** Feedback on the agile learning approaches from 2015 session

Questions	Feedback score
Overall course satisfaction	5.14
Correlation between course topics and exam requirements	5.00
Relevance for future job	5.14
Satisfaction with course workload and effort	4.86
Selection of learning activities	4.57
Teaching at sufficiently high level	4.71

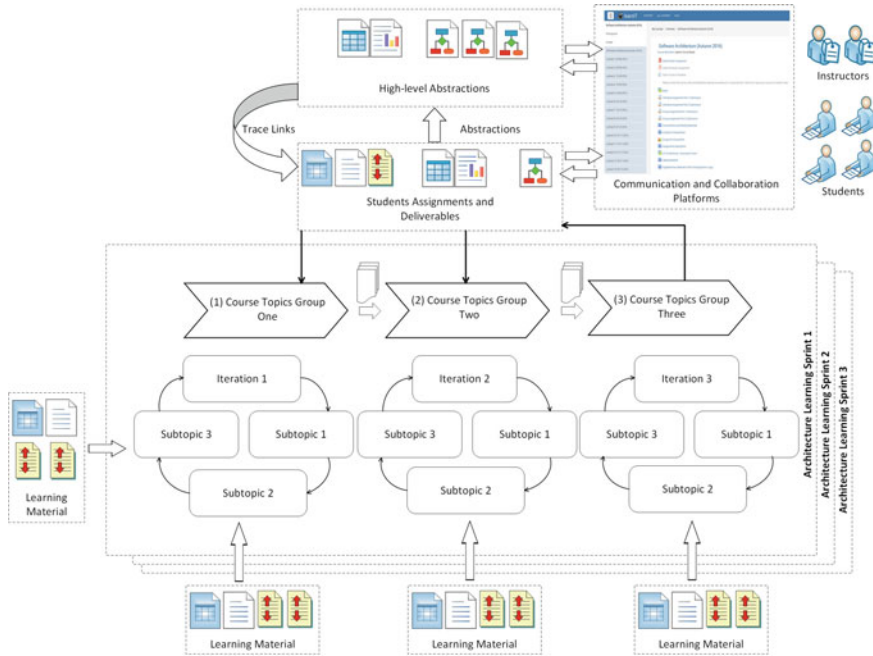
Lowest possible value = 1, highest possible value = 6, higher is better

were effective in teaching software architecture design to the students. However, it is to be noted that course evaluation was not mandatory and not all the students participated in the course evaluation.

A number of additional factors can also influence adaptation of agile education and learning in software engineering and related disciplines. The main essence of the agile education and learning approach for software engineering is to organize and deliver the course contents iteratively rather than following traditional waterfall lectures delivery approach. Engagement of the students in a learning process where they can practise applications of the agile approaches while doing class exercises and projects is also vital. Therefore, the focus of agile education methodology for software engineering disciplines should be on providing a combination of iterative delivery of the course topics during lectures combined with exercises to enable learning-by-doing.

A generic process for agile and lean learning of software engineering and related disciplines (derived from Fig. 3) is presented in Fig. 6. As the figure shows, a course following an agile learning approach can have multiple iterations on the topics during sprints. At the end of each sprint, the students should provide concrete artefacts that can be evaluated by the teaching staff members and can be used as a baseline for the next sprint. For courses that involve software development, the last iteration of each sprint can include implementation of a part of the software system. Learning material and lectures should be adjusted according to the exercises planned in each sprint. A digital education platform such as Moodle should be used so that rapid exchange of information among the students and teaching staff can take place for continuous collaboration, which is a core characteristic of every agile process. High-level abstractions can be used as a guideline for the group activities. An electronic collaboration platform can facilitate collaboration among the students, as well as between the students and teaching staff, in terms of collaborative work on the artefacts and feedback on the deliverables.

Adoption of agile education and lean learning approaches for software engineering courses can introduce additional challenges. For a course with a large number of students, managing inter-group communication and collaboration can be a challenging undertaking. A large number of students can also put additional overhead on teaching staff for providing continuous feedback on the deliverable as well as on educational institute for providing logistic support to the students so that they can practice agile.



**Fig. 6** A generic agile teaching and learning process for software engineering courses

For the courses that are of theoretical nature, agile education approaches might not be applicable for whole length of the course. Specialized courses focusing on specific ASRs including security, privacy, multi-tenancy and service level agreement compliance, might not be the best candidate to adopt agile learning approaches because a rigorous architecture analysis of whole system is needed for such ASRs before implementation can begin.

## 7 Related Work

The prospects of agile education for software architecture have not been explored much. However, a few studies have focused on agile architecture development in academic and industrial contexts. A case study on software architecting for agile projects in education has been reported in Angelov and de Beer (2017). The authors suggest dividing the achitecting activities into multiple sprints and students to be taught about the basics of agile development and software architecture in the first sprint. After that, the students can design architecture for a selected set of non-functional requirements and can participate in informal architecture reviews in each sprint. Abrahamsson et al. (2010) have discussed software architecture in the context of the agile projects and have suggested that software architecture in agile projects should focus on key

architecturally significant requirements and minimum documentation. Coplien and Bjørnvig (2011) have presented key principles of agile architecture development including deferred engineering and implementation, room for change, lightweight documentation and collaborative planning. Babar (2009) have presented common challenges faced during agile architecture development, which include incorrect prioritization of user stories and lack of focus on important architecturally significant requirements.

We have attempted to address the gaps in the related literature by providing a comprehensive approach for teaching and learning software architecture in an agile manner.

## 8 Conclusions

In this chapter, we have presented an approach for agile and lean learning. The presented approach has been developed with focus on software architecture education (tertiary level), however, the approach can be applied on other disciplines of software engineering education as well. The presented approach suggests iterative delivery of the course contents and lectures to the students followed by incremental deliverables of students' assignments and projects, and splitting students' exercises and project tasks into multiple sprints. Splitting design exercises into multiple sprints enables the students to get familiar with the agile development process and have hands-on experience with it. A lightweight software architecture evaluation process for educational environment highlights the importance of collaboration among the students for learning design of complex and large-scale systems when multiple teams work on different aspects of the same system.

We foresee that the proposed approach can help academics to align software engineering focused courses with agile practices and facilitate educational institutes to prepare their students for current and future industrial needs.

**Acknowledgements** We acknowledge the students of software architecture course in 2015 and 2016 at IT University of Copenhagen for their participation in the process and activities described in this chapter. Dr. Chauhan likes to thank his colleagues from Netcompany A/S as well, for providing valuable insight into the software architecture design challenges in agile projects.

## References

- Abrahamsson, P., Babar, M. A., & Kruchten, P. (2010). Agility and architecture: Can they coexist? *IEEE Software*, 27(2).
- Angelov, S., & de Beer, P. (2017). Designing and applying an approach to software architecting in agile projects in education. *Journal of Systems and Software*, 127, 78–90.
- Babar, M., Brown, A., & Mistrik, I. (2013). Making software architecture and agile approaches work together: Foundations and approaches. Agile software architecture: Aligning agile processes and software architectures.

- Babar, M. A. (2009). An exploratory study of architectural practices and challenges in using agile software development approaches. In *Joint Working IEEE/IFIP Conference on Software Architecture, 2009 & European Conference on Software Architecture. WICSA/ECSA 2009* (pp. 81–90). IEEE.
- Bauer, M., Boussard, M., Bui, N., De Loof, J., Magerkurth, C., Meissner, S., ..., Walewski, J. W. (2013). IoT reference architecture. In *Enabling Things to Talk* (pp. 163–211). Springer.
- Buschmann, F., Henney, K., & Schmidt, D. C. (2007). *Pattern-oriented software architecture, on patterns and pattern languages* (Vol. 5). Wiley.
- Campanelli, A. S., & Parreiras, F. S. (2015). Agile methods tailoring—A systematic literature review. *Journal of Systems and Software, 110*, 85–100.
- Chauhan, M. A., Babar, M. A., & Benatallah, B. (2016a). Architecting cloud-enabled systems: A systematic survey of challenges and solutions. *Software: Practice and Experience*.
- Chauhan, M. A., Babar, M. A., & Probst, C. W. (2016b). A process framework for designing software reference architectures for providing tools as a service. In *Product-Focused Software Process Improvement: 17th International Conference, PROFES 2016, Trondheim, Norway, 22–24 November 2016, Proceedings 17* (pp. 111–126). Springer.
- Chauhan, M. A., Babar, M. A., & Sheng, Q. Z. (2015). A reference architecture for a cloud-based tools as a service workspace. In *2015 IEEE International Conference on Services Computing (SCC)* (pp. 475–482). IEEE.
- Chauhan, M. A., Babar, M. A., & Sheng, Q. Z. (2017). A reference architecture for provisioning of tools as a service: Meta-model, ontologies and design elements. *Future Generation Computer Systems, 69*, 41–65.
- Chauhan, M. A. & Probst, C. W. (2017). Architecturally significant requirements identification, classification and change management for multi-tenant cloud-based systems. In *Requirements Engineering for Service and Cloud Computing* (pp. 181–205). Springer.
- Cleland-Huang, J. (2013). Meet Elaine: A persona-driven approach to exploring architecturally significant requirements. *IEEE Software, 30*(4), 18–21.
- Clements, P., Garlan, D., Bass, L., Stafford, J., Nord, R., Ivers, J., & Little, R. (2002). *Documenting software architectures: Views and beyond*. Pearson Education.
- Coplien, J. O. & Bjørnvig, G. (2011). *Lean architecture: For agile software development*. Wiley.
- Dingsøyr, T., Dybå, T., & Moe, N. B. (2010). *Agile software development: Current research and future directions*. Springer Science & Business Media.
- Gorton, I. (2006). *Essential software architecture*. Springer Science & Business Media.
- Hoda, R., Noble, J., & Marshall, S. (2013). Self-organizing roles on agile software development teams. *IEEE Transactions on Software Engineering, 39*(3), 422–444.
- Kazman, R., Bass, L., Webb, M., & Abowd, G. (1994). SAAM: A method for analyzing the properties of software architectures. In *Proceedings of the 16th International Conference on Software Engineering* (pp. 81–90). IEEE Computer Society Press.
- Kazman, R., Klein, M., Barbacci, M., Longstaff, T., Lipson, H., & Carriere, J. (1998). The architecture tradeoff analysis method. In *Fourth IEEE International Conference on Engineering of Complex Computer Systems, 1998. ICECCS'98. Proceedings* (pp. 68–78). IEEE.
- Kruchten, P. B. (1995). The 4 + 1 view model of architecture. *IEEE Software, 12*(6), 42–50.
- Matinlassi, M., Niemelä, E., & Dobrica, L. (2002). *Quality-driven architecture design and quality analysis method. A revolutionary initiation approach to a product line architecture*. Espoo: VTT Technical Research Centre of Finland.
- Nevo, S., & Chengalur-Smith, I. (2011). Enhancing the performance of software development virtual teams through the use of agile methods: A pilot study. In *2011 44th Hawaii International Conference on System Sciences (HICSS)* (pp. 1–10). IEEE.
- Shalloway, A., & Trotter, J. R. (2004). *Design patterns explained: A new perspective on object-oriented design*. Pearson Education.
- Sievi-Korte, O., Systä, K., & Hjelsvold, R. (2015). Global vs. local—Experiences from a distributed software project course using agile methodologies. In *Frontiers in Education Conference (FIE), 2015. 32614 2015. IEEE* (pp. 1–8). IEEE.



**Part VI**  
**Agile and Lean Activities and Games**  
**for the Classroom**

# A Practical Approach to Teaching Agile Methodologies and Principles at Tertiary Level Using Student-Centred Activities



Visham Hurbungs and Soulakshmee Devi Nagowah

**Abstract** This chapter presents a practical approach to better understand agile methodologies and principles in an educational context. An overview of the main agile principles is given along with agile methodologies that can be taught and applied in the classroom at tertiary level. The main objective is to train final year university students with agile practices currently used by the software industry. Apart from traditional activities such as homework, tests, assignments and lectures, practical approaches have been incorporated into the curriculum to engage students in active learning. In this context, the activities which were planned have been validated against agile practices at Infosys Ltd., a leading Indian-based IT company having a branch in Mauritius. In this chapter, the main focus is how team-based activities and student-centred group work have helped university students learn, understand and apply agile concepts such as Scrum, User Stories, Extreme Programming (XP), Lean, Kanban and Test-Driven Development (TDD).

**Keywords** Agile · Methodologies · Classroom · Activities · Tertiary · Learning

## 1 Introduction

The University of Mauritius (UoM) currently offers an undergraduate course in Software Engineering. 'Agile methodologies, principles and practices' are taught in the final year as a core module. The module aims at providing an in-depth understanding of agile concepts from a historical perspective. It additionally trains students on how to apply the values and principles of the agile methodologies for software development. Furthermore, students learn the skills, techniques and mindset needed to set up

---

V. Hurbungs (✉) · S. D. Nagowah  
Department of Software and Information Systems, Faculty of Information, Communication & Digital Technologies, University of Mauritius, Reduit, Mauritius  
e-mail: [v.hurbungs@uom.ac.mu](mailto:v.hurbungs@uom.ac.mu)

S. D. Nagowah  
e-mail: [s.ghurbhurrun@uom.ac.mu](mailto:s.ghurbhurrun@uom.ac.mu)

an agile team, collaborate with stakeholders and support self-organising agile teams. It has been, however, observed that UoM students have difficulties in understanding the essence of agile methodologies by simply learning the theoretical aspects of the module.

Therefore, the module map was modified to include practical approaches to agile methodologies and principles as from the academic year 2016–2017. One of the learning styles adopted for the module is team-based learning, which emphasises on students practicing course concepts to solve problems rather than simply covering content (Michaelsen & Sweet, 2008). This learning style is closely related to the concepts adopted by self-organising teams in agile methodologies. It includes important elements such as working in groups, accountability for individual and group work, frequent and timely feedback and design thinking which promote learning and team development (Dorst, 2011; Dym, Agogino, Eris, Frey, & Leifer, 2005; Michaelsen & Sweet, 2008). The new module map has inculcated in students the concept of active learning (Bonwell & Eison, 1991) by engaging them into meaningful learning activities and helping them to think about what they are doing. All students were given the opportunity to engage in active learning while participating in activities in the classroom rather than follow a traditional agile lecture where students passively received information from the lecturer.

Additionally, the new module map was designed to cater for a large number of students. A total of 80 students followed the new module map for the first time in 2016–2017 and the feedback received was positive and very encouraging. Students acknowledged that the class activities, teamwork and games helped them better understand concepts such as Scrum, User Stories, XP, Lean, Kanban and Test-Driven Development (TDD). The rest of the chapter is structured as follows: Sect. 2 describes the different agile methodologies and principles. Section 3 presents the different classroom activities and lab practicals adopted to help university students better understand the agile methodologies, principles and practices. Important feedback and observations from students with respect to the activities have also been included in Sect. 3. Feedback concerning the whole module has been included in Sect. 4. Section 5 finally concludes the chapter.

## 2 Agile Methodologies and Principles

This section describes the main agile methodologies, principles and practices used by the software industry. Agile methodologies follow the agile manifesto which aims at providing better ways for developing software (Fowler & Highsmith, 2001). Compared to the traditional methods, the manifesto presents agile principles which value more ('Manifesto for Agile Software Development', 2001):

- Individuals and interactions over processes and tools.
- Working software over comprehensive documentation.
- Customer collaboration over contract negotiation.
- Responding to change over following a plan.

In traditional methods, systems are fully specifiable, whereas in agile methodologies, the systems are mostly adaptive. Nerur, Mahapatra, and Mangalaraj (2005) highlight a number of other differences between traditional systems and agile methodologies. There is a growing interest in agile methods in industry. The two main reasons for agile adoption highlighted during a survey carried out by Version One are that agile methods accelerate product delivery and enhance the ability to manage changing priorities ('Unified Agile & DevOps', 2018).

## 2.1 Scrum Methodology

Scrum is an agile and lightweight methodology which aims at improving project productivity (Barza, Cardozo, França, Neto, & Silva, 2010; Cervone, 2011). Scrum consists of three main components: roles, activities and artefacts (Cervone, 2011). The set of predefined roles include the product owner, scrum master and a self-organising scrum team. The main roles of the product owner are to prioritise the product requirements, act as an intermediate between the development team and the business, and decide on the release date and content (Mahalakshmi & Sundararajan, 2013). The scrum master helps to remove obstacles in the team and plans daily scrum meetings. The scrum team is a self-organising team consisting of normally five to ten persons who develop the product (Cervone, 2011; Hoda, Noble, & Marshall, 2010).

The methodology makes use of an iterative and incremental approach as shown in Fig. 1. Activities in the scrum development process include sprint planning, sprint review and scrum meeting (Hoda, Noble, & Marshall, 2008). The sprint planning meeting is organised between the scrum master, the product owner and the scrum team to define the product requirements which is listed in an artefact called product backlog (Cervone, 2011). Once the meeting is over, the sprint begins. It is 'a set of development activities conducted over a predefined period, usually one to four weeks' (Schwaber, 1997). The development team normally selects some requirements from the product backlog and includes it in another artefact called the sprint backlog. A daily meeting is organised by the scrum master and the development team to enable team members to describe their tasks and concerns (Hoda et al., 2010). A sprint review is conducted at the end of each sprint to provide feedback about the previous sprint in terms of tasks achieved. Details of the next sprint are also discussed and defined in the sprint review.

Scrum lays emphasis on work done through the use of burn down charts: the sprint burn down chart for documenting the progress of the sprint, the release burn down chart for keeping track of the progress of the release, and the product burn down chart for detailing the overall project progress (Cervone, 2011). Several games have been used to teach scrum in a university environment namely Scrum Game, Scrum Simulation using LEGO Bricks, PlayScrum, Scrumia and Scrum Lego Challenge (Paasivaara, Heikkilä, Lassenius, & Toivola, 2014).



Fig. 1 Scrum methodology

## 2.2 User Stories

User stories are used in agile software development to describe a functionality that adds value to a user or customer (Cohn, 2004; Patton, 2014). O’heocha and Conboy (2010) define user stories as ‘a technique of establishing a shared understanding of software requirements using a low-overhead, user centric and flexible approach’. The user stories have the following characteristics: independent, negotiable, valuable, estimable, small and testable (Cohn, 2004).

User stories consist of 3Cs which stand for Card, Conversation and Confirmation as illustrated in Fig. 2. ‘Card’ represents a written description of the user story that can be used for planning or reminder purposes, ‘conversation’ aims at representing additional information regarding the story and ‘confirmation’ conveys the tests that should be carried out to confirm the user story is complete and working as expected (Cohn, 2004; O’heocha & Conboy, 2010). Compound and complex user stories which are generally large form an ‘epic’ that needs further decomposition (Cohn, 2004).

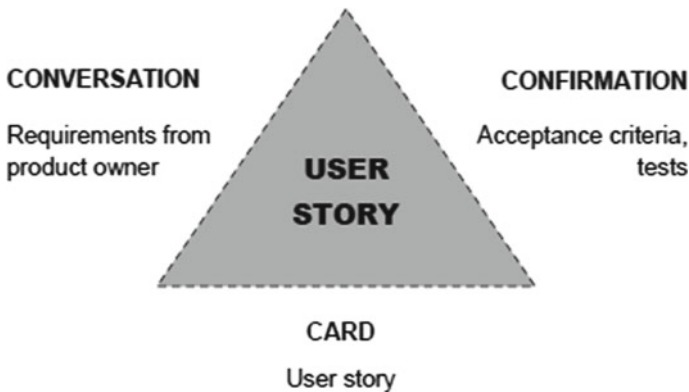


Fig. 2 User stories

### 2.3 *Extreme Programming*

Extreme programming is a well-known agile methodology guided by the agile manifesto. It lays emphasis on important values such as simplicity, communication, feedback and courage to be integrated into the software development process (Lindstrom & Jeffries, 2004). Extreme programming is a methodology explicitly built on its values and practices (Lindstrom & Jeffries, 2004). It additionally defines a set of best practices for managing the development team (Beck, 1999, 2000) as shown in Table 1.

Pair programming is one of the main concepts in extreme programming where programmers write the production codes and perform tasks in pairs (Bryant & Romero, 2006; Muller & Tichy, 2001). In a university environment, pair programming has improved student retention, confidence and program quality and is seen as a collaborative learning technique (Bryant & Romero, 2006; McDowell, Werner, Bullock, & Fernald, 2006; Muller & Tichy, 2001; Williams, Wiebe, Yang, Ferzli, & Miller, 2002). Refactoring refers to the process of improving the design of code without affecting its external behaviour. It aims to keep the design simple and maintainable (Keefe, Sheard, & Dick, 2006). Test-Driven Development (TDD) is the process of writing the test cases before the codes using unit test automation. TDD has been adopted in higher education (Kollanus & Isomöttönen, 2008) and according to Keefe et al. (2006), it is one of the most difficult XP practice to apply for students. Simple design aims at writing codes without unnecessary complexity and duplication (Astels, Miller, & Novak, 2002). Small releases refer to the development of software with a minimal useful set of functionality as the first release. Additional functionalities are then added incrementally to this first release (Beck, 1999). The Planning Game is an activity whereby customers decide on the scope and dates of the releases based on estimates submitted by the programmers who in turn decide on the next step ahead (Beck, 1999; Lindstrom & Jeffries, 2004). During the continuous integration phase, as soon as a task is completed and new codes are ready, they are integrated in the

**Table 1** Extreme programming practices

	XP practice	Short description
1	Pair programming	Work in a team of two person
2	Refactoring	Restructure existing codes
3	Test-driven development	Write tests before codes
4	Simple design	Avoid complex code structures
5	Small releases	Deliver frequently
6	Planning game	Quickly come up with a tentative plan
7	Continuous integration	Merge codes into a shared repository
8	Acceptance tests	Evaluate system compliance with requirements
9	Collective ownership	Promote collective responsibility

whole system. All tests must pass after the integration (Kuppuswami, Vivekanandan, Ramaswamy, & Rodrigues, 2003). With collective ownership, developers in a team have the right to change and improve the codes all the times and all developers take responsibility of all the codes (Lindstrom & Jeffries, 2004). The customer defines one or more acceptance tests to show that a functionality or feature is working correctly (Lindstrom & Jeffries, 2004).

## 2.4 Lean

The Lean methodology is based on concepts of Lean Management for Software Engineering (Janes & Succi, 2014). Lean development focuses mainly on removing waste from a system and improving value for the customer (Corona & Pani, 2013). The main principles adopted by Lean are listed in Table 2 (Poppendieck & Poppendieck, 2007).

According to Poppendieck and Cusumano (2012), unnecessary features, lost knowledge, partially done work, handovers and multitasking are among the biggest causes of waste in software development and lean methodology aims at eliminating such waste. The Defer Commitment principle ‘is based on the idea of making a decision at the last responsible moment or delay commitment’ (Svitis, 2013). Software should be optimised as a complete product that fits the purpose of customers, rather than just parts (Poppendieck & Cusumano, 2012; Svitis, 2013). Empowering the team aims at letting those people who add value, use their full potential to carry out a task (Poppendieck & Poppendieck, 2003). The principle also encourages the team to resolve and address their own problems (Svitis, 2013). The goal of Amplify Learning is to experiment, increase feedback based on data and incorporate things learnt to tackle tough problems that are barriers to success (Poppendieck & Poppendieck, 2003; Svitis, 2013). It is of utmost importance to produce releases frequently such as daily, weekly or even continuously to deliver fast and promote competitive advantage (Poppendieck & Cusumano, 2012). Additionally, integrity has to be incorporated in

**Table 2** Lean principles

	Lean principle	Short description
1	Eliminate waste	Remove processes that do not add value
2	Defer commitment	Make decisions using precise information
3	Optimise whole	Consider the product in its entirety
4	Team empowerment	Promote the skills of all team members
5	Amplify learning	Learn from past experiences
6	Deliver fast	Quick delivery to be more competitive
7	Build integrity in	Improve quality

software product by avoiding defects, which in turn provide high-quality software and competitive advantage (Poppendieck & Poppendieck, 2003; Svitlis, 2013).

At Oakland University, a problem-solving learning approach has been adopted to teach Lean principles to university students whereby the latter team up to conduct a Lean analysis of a real-world manufacturing system or service system (Van Til, Tracey, Sengupta, & Fliedner, 2009). Da Silva, Xambre, and Lopes (2013) adopted a game-based approach to teach professionals concepts of Lean production.

### 2.5 Kanban

One of the key Lean practices applied to the software process is Kanban (Corona & Pani, 2013; Jyothi & Rao, 2012). Similar to Lean, Kanban defines, manages and improves systems with the aim of delivering value-added services to customers (Anderson & Carmichael, 2016). Kanban makes use of a Kanban board as shown in Fig. 3 ('What is a Kanban Board?', 2018). The main activities of a particular project are highlighted on the board. The latter allows software teams to display task cards, visualise the workflow and stages of the development process, limit work in progress (WIP) at each workflow stage and measure cycle time (Ahmad, Markkula, & Oivo, 2013; Heikkilä, Paasivaara, & Lassenius, 2016). The WIP limits are specified on top of each of the board columns.

Anslow and Maurer (2015) have set an exercise whereby students were asked to create invitation cards to learn Kanban in a university environment. From the activity, students learnt that single piece flow is faster than batch and queue process.

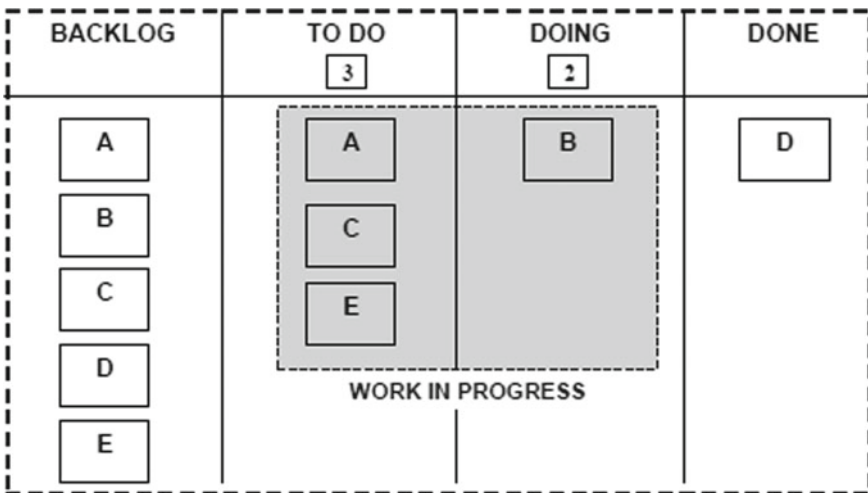


Fig. 3 Kanban board



A collaborative game such as GetKanban v4.0 has also been developed at Aalto University with the aim to teach Kanban to university students (Heikkilä et al., 2016).

## 2.6 Test-Driven Development

Test-Driven Development is a practice integrated into Extreme Programming as highlighted in Sect. 2.3. It is a testing method involving automated tests cases that are incorporated in program codes (Janzen & Saiedian, 2005). Figure 4 shows the cycle adopted for test-driven development (Famuyide, 2017). A test case is written and in case the test fails, the production codes are rewritten until the test passes (Erdogmus, Melnik, & Jeffies, 2010). Once the codes have been written, they are refactored and cleaned up to improve the design as shown in Fig. 4. Automated frameworks like JUnit are often used for automated testing ('JUnit 5', 2018).

Spacco and Pugh (2006) have proposed incentives for university students to appreciate TDD and write their test cases early. Kollanus and Isomöttönen (2008) state that 'TDD is a very demanding task for university students and it is extremely hard to implement at the same time with other technically challenging tasks'. They, therefore, recommend that TDD be integrated early in the curriculum, not too early though. Lappalainen, Itkonen, Isomöttönen, and Kollanus (2010) propose a software-based solution named ComTool to ease students test writing process for TDD. ComTool aims to reduce the technical and cognitive load for students learning TDD in an early computing curriculum.

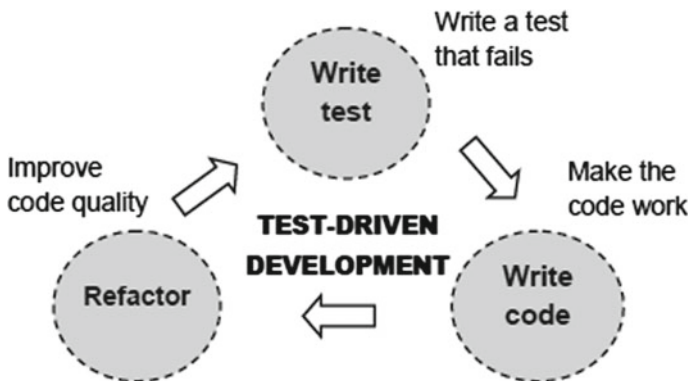


Fig. 4 Test-driven development

### 3 Practical Approaches

Agile values are difficult to teach as they represent working values (Kropp & Meier, 2013). It is therefore of utmost importance to engage students in active learning and let students adopt working styles typical to software development in small teams as practiced by the industry (Devedžić & Milenkovic, 2011). In order to achieve this, much research has been carried out in recent years. Lynch et al. (2011) define an Agile Boot Camp using a Lego-based active game to ground agile development principles. Battou (2017) presents a framework for designing an adaptive learning system based on a balanced combination of agile learner design and learner-centred approach. However, this work highlights the foundation of a framework without mentioning the practical side of agile. D’Souza and Rodrigues (2015) propose Extreme Pedagogy, an agile teaching–learning methodology for engineering education. Extreme Pedagogy has three characteristics: Learning by continuous doing, learning by continuous collaboration and Learning by continuous testing. This work describes the transition from Extreme Programming to Extreme Pedagogy. However, it does not provide any practical clues on how the practical agile objectives could be achieved. Kropp and Meier (2013) provide an overview of teaching agile software development at university level using values, management, and craftsmanship. This work presents some agile game development workshops to teach Scrum, user stories extreme programming among others. However, the detailed game development steps are not provided.

Our work, therefore, aims to bridge the gap between agile theory and practice, and presents detailed practical approaches with respect to different agile methodologies and practices in a single work. It additionally demonstrates how university students can better understand agile concepts by adopting active learning and being engaged in team-based activities similar to working styles in the industry.

The practical approaches adopted are divided into two sections as shown in Fig. 5:

1. **Classroom Activities:** The main objective of the classroom activities was to give students hands-on experience of the different agile concepts. The activities are described in Sect. 3.1. These activities not only apply to software engineering

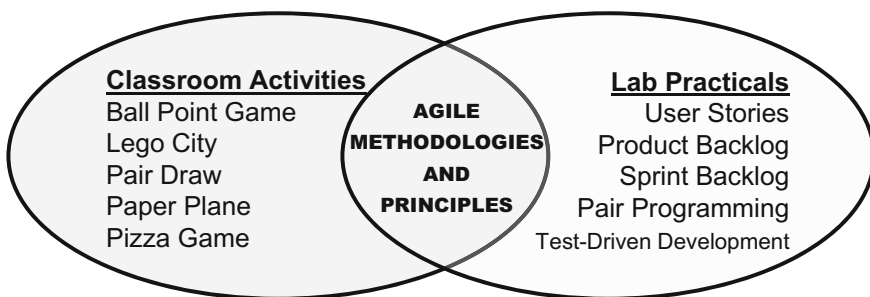


Fig. 5 Agile activities and practical approaches

but the principles could also be applied outside the context of software engineering. Since agile supports adaptive planning, quality, early delivery, continuous improvement, flexibility, customer involvement and fast response to changes among others, it could also be used for real state construction or management project (Bahceci & Holmgren, 2014).

2. **Lab Practicals:** The main objective of the lab sessions was to initiate students to agile software tools. They then applied the concepts learned from the classroom activities by using the tools to practice what they learned. The lab practicals are described in Sect. 3.2.

### 3.1 Classroom Activities

The student-centred classroom activities that were brought together to explain the agile methodologies and principles in a tertiary educational context are detailed in Table 3.

The above classroom activities are described in more details in the next subsections along with steps used, outputs from students, images, formula used and feedback from students. The main objective of each activity is also discussed in terms of agile teaching and learning.

**Table 3** Agile classroom activities

	Agile methodologies and principles	Activities	Objectives
1	Agile core values	Ball Point Game	Show the importance of both teamwork and individual skills
2	Scrum	Lego City	Use Lego blocks to help students translate user stories into backlog tasks and prioritise tasks
3	Extreme programming	Pair Draw	Help students understand the benefits of working as a pair
4	Lean	Paper Plane	Make students determine their group productivity and eliminate waste during production
5	Kanban	Pizza Game	Help students know how to minimise work in progress and maximise on completed tasks and visual workflow

### 3.1.1 Ball Point Game

The first objective of the module was to inculcate students with basic agile principles. One example is to form a big team and demonstrate that success depends on the team output, not on individual output. A variation of the Ball Point Game by Boris Gloger was set up (Boersema, 2011; Waters, 2011). In this variation, ping-pong balls had to be thrown in cups. Students organised themselves in teams of 4–5 members of different abilities and each team was given a total of 6 balls. The team members were not predetermined but students chose their teams randomly before starting the activity. Team members had to throw a maximum number of balls in the cups placed on the table. Each team was also responsible to self-organise itself and allocate tasks to members based on individual skills. This was achieved via several rounds where each team had the opportunity to produce a better output after each round.

**Activity Requirements**

*Ping-pong balls, Plastic cups, Water bottle, Desks or appropriate space to throw the balls.*

For this experiment, a total of three cups were used and the time limit was set to 1 min, although these are not strict guidelines. The parameters could be changed based on the team size and time available for the activity. If the ball entered the cup, a counter was incremented and the ball was re-used as part of the activity until the round was over. The Ball Point Game setup and illustration are shown in Figs. 6 and 7.



**Fig. 6** Ball point game activity setup



**Fig. 7** Ball point game illustration

At the end of the time limit, the total number of balls successfully entered in a cup was counted. The activity was repeated over a minimum of 2 rounds. It is also important to note that teams were allowed to choose their own strategy to maximise on the number of balls. If in the first round, the strategy was not successful, the team had the choice to change strategy and re-organise itself in subsequent rounds. Sample results for two teams over 3 rounds are summarised in Table 4.

From the sample results obtained, it is clear that the teams struggled in the first round to have a maximum number of balls since they did not know who was good at throwing balls, what the best strategy to adopt was and how to synchronise the team members. In round 2, the teams learnt from their mistakes and were able to throw more balls in the cups. One strategy that was adopted was throwing and picking of balls in parallel by selected team members in order to gain on time. In round 3, the ball-throwing process improved as the team was fully functional, team members were allocated tasks for which they were skilled at, and there was good communication between the members. One strategy that was adopted in this round was an allocation of tasks based on skills. As an example, students who were good at throwing balls were assigned the ball-throwing task while others were picking the balls. To summarise, the teams were able to throw more balls in the cups in the final round compared to round 1 or round 2.

**Table 4** Ball point game results sample 1

	Team A	Team B
Round 1	3	5
Round 2	5	9
Round 3	7	13

Feedback was collected from each team after the experiment. The feedback proved that the objectives set for this activity were achieved and students understood the basic agile principles. Some of the feedback is listed below:

1. *The game relied on close co-operation and collaboration between all team members in order to gain the highest score.*
2. *Every team member must share ideas and brainstorm about the different possibilities to put maximum balls in the cup.*
3. *We have to assign team members the task which they have more knowledge on, like aiming to put the balls in the cups more often.*
4. *After each round, we had to discuss and improve on what we previously did.*
5. *Time limit should be respected and other rules as well like distance from the table and the person.*
6. *Working as a team was crucial to win the game.*
7. *Analyse the team member's ability and assign them to what they do best so as to increase team efficiency and get the job done quicker.*
8. *We have understood that it was really important to learn from our mistakes and those from others in order to move forward.*
9. *We learned that we have to organise ourselves based on our skills in a team to be effective and saw the direct implication through the activity.*
10. *We needed to be flexible. At a given time, we had to swap the tasks when a particular team member was not able to perform as expected.*

After the Ball Point Game, students recognised the importance of self-organising teams, learned from previous experience and they were given the opportunity of deciding on the best strategy to complete a group work. Task allocation based on skills, time management and the significance of team collaboration was also part of this activity.

One of the biggest challenges of this activity was team collaboration. Since the team members were chosen randomly, each team consisted of team members of different abilities. In the first round, it was observed that some members could not collaborate together since it was the first time they were working together. There was also a lack of leadership in some teams. As part of the strategy to be adopted, the teams had to appoint a team leader who would guide them during the activity and assign tasks based on individual skills. Since some students were working together for the first time, it was challenging for the team leader to assign tasks appropriately.

### **3.1.2 Lego City**

The objective of this experiment was to help students understand the Scrum methodology and learn how to write requirements in terms of user stories. The Lego City

(Friesen, 2015) was chosen as the class activity where the team had to put into practice different aspects of Scrum such as user stories, product backlog, prioritisation and sprints among others. From a training perspective, students were expected to experience the role of the following stakeholders: Product Owner, Team member and Scrum master.

### **Activity Requirements**

*List of user stories with acceptance criteria, Lego blocks representing city element, Bristol papers, Coloured pencils, Markers and Rulers.*

The Lego City steps are as follows:

1. Grouping of students in different teams of approximately 5 students to inculcate concepts such as self-organising teams.
2. Each team received a set of user stories which they had to translate into feasible tasks. A stand-up meeting was conducted before each sprint to discuss on the way forward.
3. The team then implemented the tasks and constructed different parts of the city in several rounds.
4. During implementation, some additional user stories were given to selected team(s) to make students learn how to deal with changing requirements.
5. At the end of the activity, the teams had to merge their work to form the final Lego City.
6. Finally, the teams had to review the Lego city and discuss on implementation issues. This step helped them better understand the concept of retrospective.

User stories were given to students using the user story 3Cs format: Card, Conversation and Confirmation. The Lego City activity was structured in terms of Epic, Theme and User Stories as shown in Fig. 8.

The main focus of this activity was to make students learn the scrum iterative framework by building a city using Lego blocks and user stories. The latter were pre-compiled and represented various buildings and elements of the city. In this activity, the city consisted of different themes (road network, animal park, school, consulate and construction site) as shown in Fig. 9. These themes were further broken down into user stories that represented the requirements of the city. Each team was allocated a theme and the corresponding user stories which were then converted into feasible tasks. Teams had to complete each task in different sprints. One team was also given additional user stories during one of their sprint. They had to decide on how to proceed with the new user stories and had to judge whether the new stories were of higher priority. If the priority was higher, these stories were then allocated in the next sprint.

Figure 9 illustrates the completed Lego City after all teams have merged their tasks. A video version of the Lego City activity is also available on YouTube on the following link: <https://youtu.be/AKjIDZyfGic>.

EPIC Lego City				
THEME Road Network	THEME Animal Park	THEME School	THEME Consulate	THEME Construction Site
STORY Roundabout	STORY River	STORY Building	STORY Building	STORY Site
STORY Junction	STORY Fencing	STORY Environment	STORY Security Guard	STORY Trucks
STORY Road Signs	STORY Animals	STORY People	STORY Car port	STORY Workers
STORY Traffic	STORY Trees			STORY Materials
STORY Vehicles				

Fig. 8 Lego city activity breakdown



Fig. 9 Scrum Lego city



Similar to the first activity, feedback was collected from students. Some of the feedback received are listed below:

1. *We learned the importance of planning beforehand so as to better organise the upcoming task.*
2. *Absence of a teammate affected the work. We learnt how to manage the work without 'Abdel'.*
3. *Lack of collaboration with external teams. Better take into consideration the work of other teams which might affect our work.*
4. *Distributing tasks to each member according to his/her capacity helped to give a better and faster result. Each member proposed his/her ideas, and all the ideas were combined to build the project. Changes were constantly being done and monitored to give a better result.*
5. *The game has helped us to collaborate among each other towards a common success and the end result that is the city element was built as per expectations.*
6. *With short sprints and constant feedback from the team, it was easier to cope with the changes needed.*
7. *The Lego city taught us that bringing our skills together and with a good design decided by the team made us aware how important it is to cooperate and make a good project with dynamic communication between members.*
8. *Face-to-face conversation is mostly used to discuss issues with team members.*
9. *We had to manage our task in a time frame and assign our resources in an efficient way. Moreover, more requirements were added in the middle of the time frame and we had to adjust everything in a limited amount of time.*

After the Lego experiment, students learned how to build a product backlog and technical tasks from user stories. They also practiced concepts such as sprints, stand-up meetings, teamwork and face-to-face communication to reach a common goal related to the Scrum methodology.

Students faced several challenges for the completion of the Lego City. First of all, the Scrum methodology was new to them and they had some difficulties making the transition from the waterfall style of product development to agile development. They took some time to practice the iteration process as some teams started constructing their Lego blocks in the first step. Students had to be reminded that they had to use Scrum concepts such as Sprint planning, Sprint review, Sprint retrospective in each iteration for better task organisation. Another challenge encountered was lack of collaboration with external teams since each team was working in isolation. To tackle this issue, the Scrum Master should ensure that all teams are well coordinated by enforcing Scrum values and practices. Last but not least, the precompiled user stories should be well defined since they would affect the output of each team. The

user story must be well written including the acceptance criteria so that all team members have a common idea of the requirements.

### 3.1.3 Pair Draw

The objective of the Pair Draw activity (Kerievsky, 2001) was to help students understand the concept of Extreme Programming. In this experiment, students worked in pairs, checking each other’s work and providing the support to do a good job. They were given the task of drawing one of their friends. Figures 10 and 11 illustrate some of the sample individual and pair drawings that students had drawn as part of this activity.

**Activity Requirements**  
*A4 paper, Pencils or Pens with at least 2 different colours.*

The Pair Draw steps are as follows:

- **Round 1:** Each student draws an image of one of their friends.
- **Round 2:** Students pair themselves and draw a single image of one of their friends together in only one drawing. One student draws the left-hand side while the other student draws the right-hand side of the drawing at the same time.
- **Round 3:** Students change pair and re-draw a single image of another friend together in only one drawing. Again, one student draws the left-hand side while the other student draws the right-hand side of the drawing at the same time.



Fig. 10 Sample drawings by individual students



**Fig. 11** Sample drawings by a pair of students

Some observations gathered after the Pair Draw activity are listed below:

1. *In step 1, students completed their drawings without any interaction with other students.*
2. *In step 2, students were able to discuss on the way to proceed with their drawings. Those who were more skilled at drawings led the team while the other student followed the same path. In addition, any errors made by one student was immediately rectified by his/her pair. The quality of the drawing was also continuously monitored in pair.*
3. *In step 3, when the pair changes, not every pair could come up with the same quality drawing as in Step 2. Students had to adapt to their pair and the new pair generated different drawings.*

After the Pair Draw activity, students learned the importance of collective ownership and avoid long working hours by using the concept of pair programming. Collective ownership allowed the students to take responsibility for the all the work done by the pair. If the work is of good quality, both students got credit for their work. Otherwise, both of them were accountable if the work was of poor quality.

Students were very much involved in this activity as they were drawing images of their friends and this made the activity more interesting. In round 2, students selected one of their friends to work in pair and some of them came up with very good drawings. However, in round 3 when the pair changed, the quality of the images was not the same. Those who were able to draw good images in round 2 could not achieve the same quality drawing when working in a different pair. Therefore, it

is important to highlight that it is the skills of both students in a pair that would determine the quality of the output.

### 3.1.4 Paper Plane

The objective of the Lean Paper Plane activity (Boersema, 2012) was to teach the basic principles of the Lean methodology to students. Each group of students had to assemble paper planes without defects. Only planes without defects were counted as good planes and the activity was repeated more than once. The process was iterated twice for a period of 5 min.

#### *Activity Requirements*

*A4 white paper, Ruler and Pen or pencil.*

In this experiment, the students practiced the following Lean principles:

- Eliminate waste and defects.
- Learn as the product is being developed.
- Make decisions on correct information.
- Empower the team by involving all team members in decision making.
- Fast delivery of completed works and minimise partially completed work.
- Optimise the entire product, not just parts of the product.

The following productivity formula ('Overall Equipment Effectiveness', 2017) was used to evaluate the correctness of the paper planes of each team:

$$\text{Productivity} = (\text{Good Count} * \text{Ideal Cycle Time}) / \text{Planned Production Time}$$

#### **Example**

5 min and 2 good planes

*[Assuming ideal cycle time to build a good plane is approximately 1 min]*


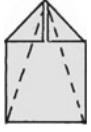

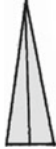
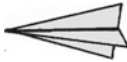
$$\text{Productivity} = 2 * 1/5 = 0.4 \{40\%$$

Ideal situation: 5 good count, 100% productive.

The assembly of the paper plane includes the steps as shown in Table 5.

The students grouped themselves in teams of 4–5 students for this activity and the productivity of each team was computed at the end of each round, which was of 5 min duration. The assumption was that one paper plane assembly took on average 1 min. Each team improved their productivity over time and they learned how to produce planes with less defects. Teams which adopted the Lean concepts normally had more planes without defects. Sample results of 2 teams over 2 rounds are shown in Table 6.

**Table 5** Assembly of the paper plane (Boersema, 2012)

Step	Description	Illustration
1	Fold first corner on one side	
2	Fold first corner on each side	
3	Fold wing on each side	
4	Fold wing on each side again	
5	Finished paper plane	

**Table 6** Lean paper plane productivity

	Round 1 productivity	Round 2 productivity
Team A	$2 \times 1/5 = 40\%$	$3 \times 1/5 = 60\%$
Team B	$0 \times 1/5 = 0\%$	$1 \times 1/5 = 20\%$

Some of the lessons learned from this activity are:

1. *One mistake students made was wastage and overuse of resources.*
2. *In the first round, students rushed towards the completion of a maximum number of planes without considering quality. Many planes, therefore, contained defects.*
3. *The planes that were not fit for use were discarded and therefore considered as wastage of resources.*
4. *In round 2, each team recognised the importance of quality and they produced better planes with less defects and were, therefore, more productive.*

After the paper plane activity, students learned how the removal of waste from a production system improved the product value for the customer. Students also determined their productivity as a team and they acknowledged that good team cohesion resulted in better decision making, team empowerment, fast delivery and product optimisation.

The first challenge of this activity was that students had difficulty in achieving a perfect paper plane in round 1. They ignored the quality aspect of the paper planes and almost all teams started creating a maximum number of paper planes. The first round resulted in a lot of paper wastage and therefore, adequate provision must be made in terms of the number of papers provided to each team. In round 2, defects in the paper planes were removed and teams were now aware that they should focus on quality. The second challenge was that the quality parameters and defects should be well defined before round 2 so that teams are aware of the expected output. For example, paper planes should be symmetrical and folds should be done neatly. It was observed that teams had varying definitions of the paper plane quality.

### 3.1.5 Pizza Game

The objective of the Pizza Game (Swanson, 2013) was to help students learn the basic principles of the Kanban methodology in terms of product development with an emphasis on continuous delivery while not overloading the development team. This activity concentrated on the visual representation of work items and allowed team members to see the state of a particular piece of work at any point in time. A kanban board with task decomposition was used as part of the Pizza Game. The board enabled the team to remain focused and concentrate on the cycle time of delivered work and resolve bottlenecks by providing a visual view of the workflow.

#### **Activity Requirements**

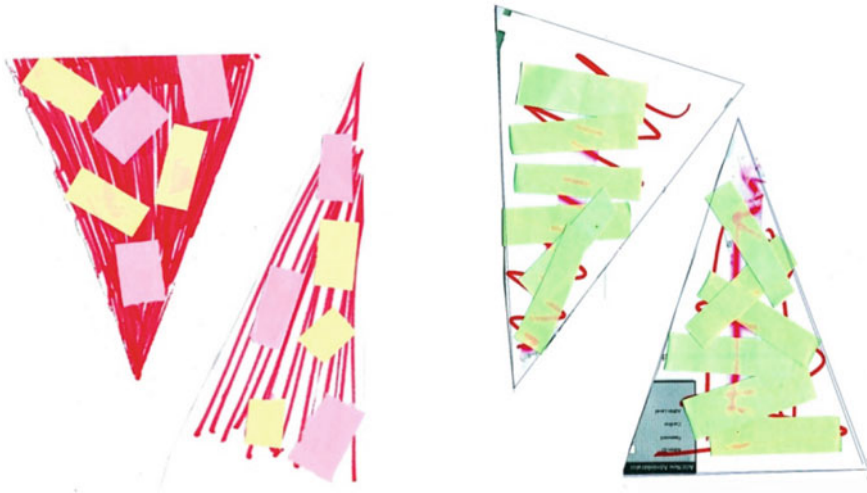
*A4 coloured paper (White, Yellow, Pink, Green), Red markers, Paper Glue or Double-sided tape, Scissors and Bristol paper.*

For this experiment, students had to group themselves in teams and they were given the materials to build the pizza. Sample pizza slices are illustrated in Fig. 12. The steps of the Pizza Game are detailed in Table 7.

After each round, the output of the teams was measured. The formula used was the sum of finished slices and remaining pizza bases and toppings. Table 8 shows the results of 2 teams over 3 rounds.

$$\text{Output} = \text{Finishedslice} + \text{Pizza base} + \text{Toppings}$$

From the results, it is clear that both teams improved the pizza delivery process. In round 1, the teams concentrated more on the final pizza slices without paying



**Fig. 12** Sample Kanban Pizza slices

attention to the workflow or work in progress. There was a wastage of items and process policies were not well defined. However, the output was superior for both teams in round 3 compared to round 2 and round 1. This was possible by taking into consideration kanban principles such as workflow visualisation and limit work in progress during the pizza-making process. A sample kanban workflow based on the pizza activity is shown in Fig. 13.

Similar to the previous activities, feedback from students was recorded. Some of the feedback received are listed below:

1. *Lessons were learned by making mistakes. One of the mistakes we made was wastage and overuse of resources.*
2. *The use of the Kanban board helped us to devise a strategy to tackle the production of the products (pizza) as we gradually learnt that the product quality depends on how well we devise a plan/strategy to carry out the activities while adding maximum value to the final product.*
3. *The activity helped in having a better understanding of Kanban. It helped us apply the concept and thus reduce our waste products and increase the final product outcome.*
4. *Doing this activity really helped us in seeing the amount of waste produced and how it was consequently reduced to obtain more deliverable products, this would have been more difficult to visualise only in theory.*
5. *We have also understood the effects of limiting our work in progress and on top of that, we have had lots of fun during the Pizza Game.*

**Table 7** Kanban Pizza Game steps ('Kanban Pizza Game', 2013)

Round	Steps	Observations
1	<ul style="list-style-type: none"> <li>• Produce as many slices until the team is told to stop</li> <li>• Workflow: Create base, Add toppings, Put in oven</li> <li>• Measure the output. Each piece counts negative until the slice is 100% done                             <ul style="list-style-type: none"> <li>– Pizza base (with or without sauce): -4 points</li> <li>– Toppings: -1 point each</li> <li>– Finished slice: +10 point</li> </ul> </li> </ul>	<ul style="list-style-type: none"> <li>• Make the workflow explicit</li> <li>• Limit the work in process (WIP)</li> <li>• Use the best pizzas to create a common definition of done</li> </ul>
2	<ul style="list-style-type: none"> <li>• Play again and repeat the steps in Round 1</li> <li>• Introduce a new pizza type where the toppings must be added after baking the pizza</li> <li>• Measure the output</li> </ul>	<ul style="list-style-type: none"> <li>• Points are given only for fully delivered orders</li> </ul>
3	<ul style="list-style-type: none"> <li>• Round 1 can be repeated several times and the output is measured</li> </ul>	
4	<ul style="list-style-type: none"> <li>• Wrap-up: Draw your workflow</li> <li>• Look back at the game</li> <li>• Draw the flow including WIP limits</li> <li>• Use the materials to make it look nice</li> </ul>	<ul style="list-style-type: none"> <li>• Six core Kanban practices: (1) visualise the workflow, (2) limit work in progress (WIP), (3) measure and optimise flow, (4) make process policies explicit, (5) implement feedback loops, (6) improve collaboratively, improve experimentally</li> </ul>

**Table 8** Pizza Game results

	Round 1 output	Round 2 output	Round 3 output
Team A	$30 - 8 - 9 = 13$	$130 - 4 - 3 = 123$	$180 + 0 + 0 = 180$
Team B	$70 - 8 - 4 = 58$	$120 + 0 + 0 = 120$	$180 + 0 + 0 = 180$

After the pizza game, students learned the importance of limiting work in process, continuous workflow, reducing waste and promoting visual control to increase throughput. They also learned how to pull individual work requests through a sequence of value-added activities, quickly and without interruption, and manage their time as a team by reallocate unused time for other tasks.

The main challenge of this activity was to make team members visualise the workflow during the activity. However, since the kanban board was mounted after the activity, it was difficult for the teams to reflect on the processes involved in pizza-making. To better help teams in this activity, it is important to give a sample kanban board prior to start this activity. This sample workflow should also highlight the idea of limiting work in process. Another challenge was that teams could not readily identify and cope with bottlenecks. Teams should, therefore, be aware of how WIP limits drive and change the behaviour of people by producing the right things and



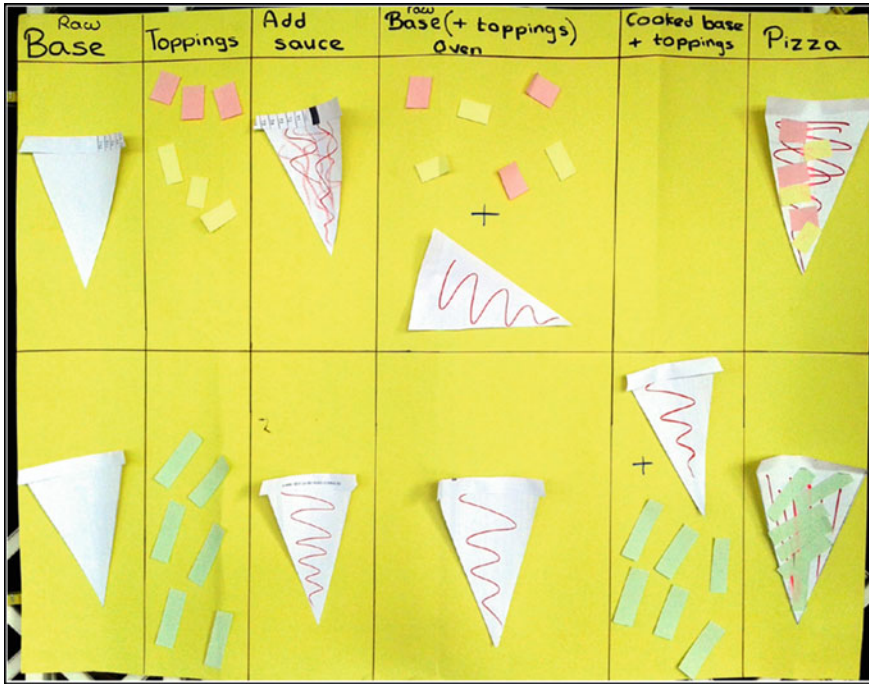


Fig. 13 Kanban Pizza workflow visual representation

avoiding wastage of materials. Yet, another challenge was that teams were different and some of them were working too fast which in turn led to the making of ugly pizzas. Therefore, it is important to make the process explicit and highlight the quality aspect to all teams and ask them to agree on the quality level prior to the game.

### 3.2 Lab Activities

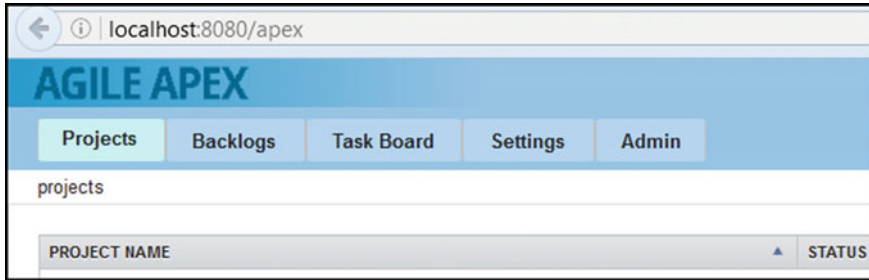
The lab activities that were designed to initiate students to agile software tools are detailed in Table 9.

#### 3.2.1 User Stories, Product Backlog, Sprint Backlog

AgileApex (AgileApex, 2017; Knepper, 2017) was chosen as the software tool for the lab practicals on user stories, product backlog and sprint backlog. AgileApex is an easy to use agile project management tool. The main features are: User authorisation

**Table 9** Agile lab activities

	Agile methodologies and principles	Activities	Software tool	Outcome
1	Scrum	User stories	AgileApex	<ul style="list-style-type: none"> <li>• Learn how to formulate good user stories using the 3 C's card format</li> <li>• Write user stories that are clear to both the product owner and the development team</li> </ul>
2	Scrum	Product backlog	AgileApex	<ul style="list-style-type: none"> <li>• Practice how to translate user stories from a real-world scenario into a set of technical tasks for the product backlog</li> <li>• Learn how the Scrum team performs backlog prioritisation by getting constant feedback from the product owner</li> </ul>
3	Scrum	Sprint backlog	AgileApex	<ul style="list-style-type: none"> <li>• Learn how to generate a sprint backlog from the product backlog during the sprint planning process</li> <li>• Learn how to monitor the sprint using a sprint burndown chart</li> </ul>
4	Extreme programming	Pair programming	Collabedit	<ul style="list-style-type: none"> <li>• Perform some Java programming tasks as a pair</li> <li>• Practice the roles of both a Driver and Observer, and learn how switching roles can affect the programming task</li> </ul>
5	Kanban	Kanban board	Trello	<ul style="list-style-type: none"> <li>• Learn how to visually monitor the work progress using a kanban board</li> <li>• Understand how kanban can be used to organise many areas of an organisation by using work on progress limits</li> </ul>
6	Test-driven development	Case study	Eclipse	<ul style="list-style-type: none"> <li>• Understand how TDD differs from traditional testing</li> <li>• Recognise the added value of TDD on code design</li> <li>• Understand the relationship between TDD and Refactoring</li> </ul>



**Fig. 14** Agile Apex (AgileApex, 2017)

and management, Product backlog, Hierarchical backlogs, Projects-Releases-Sprints and Sprint planning.

At the time of designing the practicals, AgileApex was available as a free software download. The software was set up using XAMPP which is an easy to install Apache distribution. XAMPP services used were Apache, MySQL and Tomcat. AgileApex was installed on a local server in the lab.

As shown in Fig. 14, AgileApex provides the following Scrum-related options to the user:

- **Projects:** One project can contain multiple releases and one release can contain multiple sprints.
- **Backlogs:** Product backlog can be managed by project managers or sprint planners and tasks can be re-ordered. Tasks can also be logically organised in terms of hierarchical backlogs.
- **Task Board:** Team members can edit and report their progress in the Task board page. Managers, product owners and other interest groups can also easily view progress in the Task board page.
- **Admin:** Admin can create user profiles (Viewer, Reporter, Sprint planner, Manager) with different access rights.

### User Stories/Product Backlog

In this practical, students learned how to write the requirements as user stories and translate them into a set of product backlog items on Agile Apex. Students also learned the importance of prioritising the product backlog. Each team had to perform a small brainstorming session to describe the requirements of ‘An electronic mail application’.

As shown in Fig. 15, the project manager can add tasks into the Project Backlog after the project is created. The effort for each task may represent different units of measurement depending on the project. A person who will be in charge of this task can be assigned by Editing the task at a later stage.

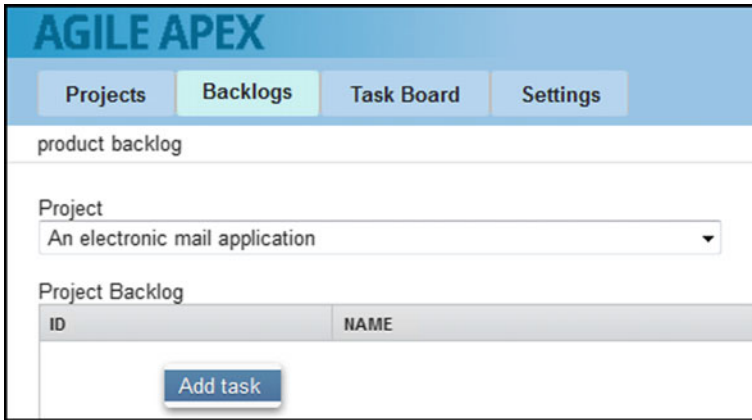


Fig. 15 Agile Apex product backlog (AgileApex, 2017)

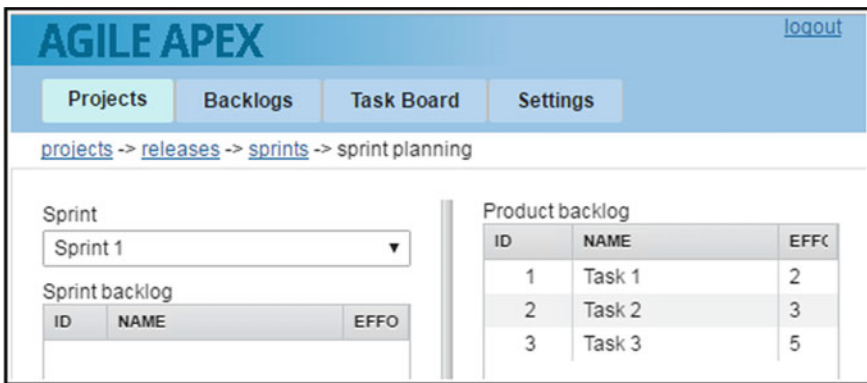


Fig. 16 Agile Apex Sprint Backlog (AgileApex, 2017)

### Sprint Backlog

The objective of this practical was to show students how to create a sprint backlog by selecting items from the product backlog. Each team was required to create the product backlog and sprint backlog for the Lego City activity described in Sect. 3.1.2 in AgileApex.

Figure 16 illustrates the Product Backlog and Sprint Backlog. During the sprint planning meeting, the team selected some number of product backlog items to form part of the Sprint backlog. Managers could drag and drop tasks between the product backlog and the sprint backlog.



Fig. 17 Collabedit Pair Programming editor ('Collabedit', 2016)

### 3.2.2 Pair Programming

Students were given some pair programming tasks using the Java language. They used the online pair programming editor, collabedit, to work as a pair for this programming exercise ('Collabedit', 2016). Collabedit is an online code editor that lets people collaborate in real time. It works in the web browser and therefore does not require any installation. This practical helped the students to understand the concept of working as a pair.

Figure 17 illustrates the Collabedit interface which provided the pair programming environment. On the left-hand side, the codes are displayed while the programming language and collaborators are displayed on the right-hand side. As collaborators are working together, everyone will see the updated codes.

### 3.2.3 Kanban Board

Students used Trello ('Buildbettersoftware—Trello Guide', 2018) as a project management tool. Trello uses the kanban paradigm for managing projects which are represented by boards, which contain task lists and tasks. Tasks can progress from one list to another and grouping of tasks is also supported. Trello is also available for iPhone, Android and Windows 8 mobile platforms.

Students were required to create a kanban board by specifying the user stories as their backlog. They prioritised and planned which user stories they would start to

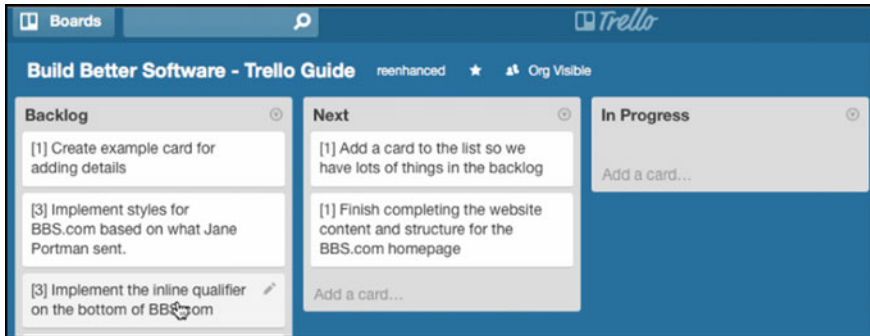


Fig. 18 Trello project management tool ('Buildbettersoftware—Trello Guide', 2018)

work with. They then practiced pulling one task from one section to another while taking into consideration work in progress. Tasks were tagged as done after they were hundred percent completed. At the end of this practical, students learned how to use boards, lists, and cards to organise and prioritise tasks.

Figure 18 illustrates the Trello user interface where the workflow can be visualised from left to right until the tasks are completed. The columns can be user-defined and adapted for each development environment. Individual tasks are represented using simple cards where the following can be defined: name of task, name of developer, due date or any other relevant information. Also, another important point is that Trello can limit assigned tasks to a reasonable number that will not produce bottlenecks.

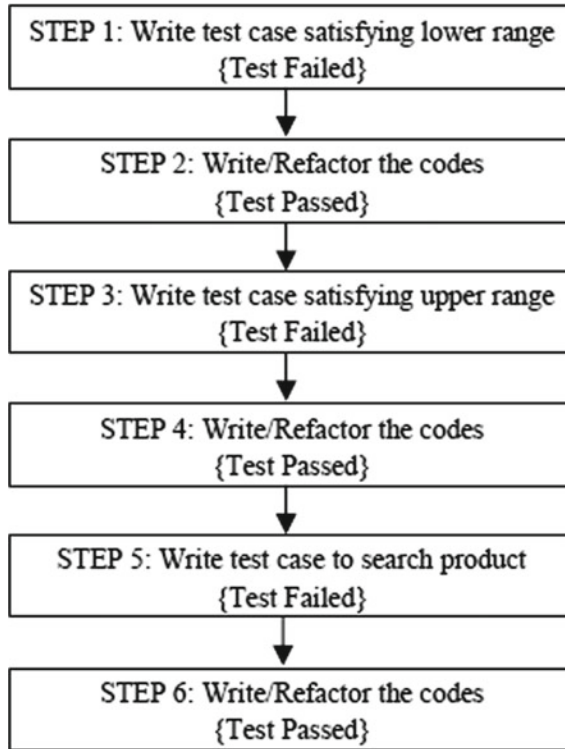
### 3.2.4 Test-Driven Development

The goal of this activity was to compare and evaluate the impact of Test-Driven Development (TDD) with Traditional Development taking into account factors related to software quality assurance and productivity. The students had to read a case study entitled 'Impact of Using Test-Driven Development: A Case Study' (Yenduri & Perkins, 2006). They use the Eclipse IDE to implement a Java application with relevant methods with data saved in an appropriate database and used the JUnit framework ('JUnit 5', 2018) for automated testing.

Students also had to apply both Traditional and TDD for testing the Java program. The following metrics were used during the testing process:

- Number of test cases written
- Number of faults detected during all the units tested
- Number of faults detected during integration testing
- Total Person Hours spent.

A sample Java program developed included the following components:



**Fig. 19** Test-driven development steps

1. **Insert:** Enter product details, perform validation on barcode digit and save the data in a database.
2. **Update:** Search product, Modify fields in the database.
3. **Delete:** Search product, display product details, Confirm product deletion.
4. **Near Depletion:** Allow the user to enter a threshold value for product quantity, Retrieve products falling within a depletion range.
5. **Near Expiration:** Allow the user to enter a threshold expiration date, retrieve products based on the expiration data.

The TDD steps used to develop the *Insert* component are shown in Fig. 19.

The overall results after testing all the components are shown in Tables 10 and 11. The test-driven approach is more productive as it resulted in fewer faults and less time taken during development. While the traditional approach took more time and produced more test errors, the value gap between both approaches was not significant. This may be explained by the test subjects were more proficient in the traditional approach compared to test-driven which was a new concept introduced in this module.

**Table 10** Traditional approach results

Module name	Number of faults detected	Amount of time taken
Insert	24	2 h 23 min
Update	14	30 min
Delete	6	2 h 28 min
Query depletion	2	42 min
Query expiration	17	2 h 30 min
Total	63	8 h 33 min

**Table 11** Test-driven approach results

Module name	Number of test cases written	Number of faults detected	Amount of time taken
Insert	28	14	2 h 33 min
Update	9	10	45 min
Delete	5	10	1 h 16 min
Query depletion	4	9	53 min
Query expiration	3	13	2 h
Total	49	57	7 h 27 min

## 4 Student Feedback

As part of the Quality Assurance process at the University of Mauritius, some of the raw feedback obtained from students after completion of the agile module in the academic year 2016–2017 are listed below in Table 12. The feedback obtained was very encouraging. It additionally supports the practical approaches adopted to build the conceptual understanding of agile concepts by engaging students in active learning. The feedback has been taken in due consideration for improving the module in terms of teaching, delivery and contents for the next academic year.

## 5 Conclusion

This chapter presents a new approach undertaken at the University of Mauritius to help final year Software Engineering students better understand agile methodologies and principles. For the academic year 2016–2017, the module map was modified to include practical approaches based on agile methodologies and concepts such as Scrum, User Stories, Extreme Programming (XP), Lean, Kanban and Test-Driven Development (TDD). Compared to other research work carried out, the chapter aims to bridge the gap between agile theory and practice and presents detailed practical approaches with respect to different agile methodologies and practices in a single work.



**Table 12** Student feedback at the University of Mauritius, 2016–2017***What did you like most about this module and teaching?***

- *Very interesting. Adapted to work environment*
- *Activities done in class brings a lot more fun in learning this module. Lecturer always encourages students to participate in class*
- *The guest lecture from Infosys illustrates what is happening in the real world*
- *How can be Agile be applied in real life project*
- *The concept of class activities enable us to have a glimpse of the real life in industry*
- *The activities demonstrating how team members should behave in a working environment*
- *Class activities are very nice. This is probably the best classes i have attended*
- *Lecturer encouraged learning through activities that enabled us to put theory to practice*
- *The module involves interaction and the lecturer provides many games and tasks to perform in groups to get the idea behind each principle in the module*

***Improvements for the module in terms of teaching, delivery and contents***

- *Too many methodologies (Agile, Scrum, Kanban) resemble each other, so it can be confusing at times. Maybe, have a summary of each methodology at the beginning of each week just to really differentiate them*
- *The lecture notes could have included some more details about each agile practices*
- *Lecturer must continue using interactive tutoring and live talk from experts*
- *The lecturer can give more feedback to students for them to ensure that they are on the right track*

***Any other relevant comments***

- *Hats off to the lecturer who was able to turn a pretty boring theory class into a really interesting one!*
- *In overall, turn out that activities put in place really gave me a view of Agile if I had read only the lecture. I think these activities make answering case studies much easier. Gives you a broader overview of the approaches*

Moreover, learning styles adopted for the module were active learning and team-based learning which emphasised on students practicing course concepts to solve problems rather than simply learning the theory. After each activity, feedback was gathered from students and relevant observations were also recorded. Students enjoyed activities like playing ping-pong, constructing Lego blocks, building paper planes and playing pizza game while learning at the same time. Students greatly valued opportunities to engage themselves in team-based activities and practical experiments for this module. They viewed this as a means to help them better understand agile concepts and maintain their interest during the sessions. The activities encouraged all students to participate with high enthusiasm that enhanced their learning process. Additionally, they learnt to work as a team, understand the importance of teamwork and recognise the different roles within a team consisting of members with different skills.

**Acknowledgements** We would like to thank all the students of B.Sc. (Hons) Software Engineering Level 3 (both normal cohort and mixed-mode), who participated and provided their valuable input for the experiments during the University of Mauritius academic year 2016–2017.

## References

- AgileApex [Computer software]. (2017). Retrieved from <http://www.agileapex.com>.
- Ahmad, M., Markkula, J., & Oivo, M. (2013). Kanban in software development: A systematic literature review. In *39th Euromicro Conference on Software Engineering and Advanced Applications*. Anderson, D., & Carmichael, A. (2016). *Essential Kanban condensed*. Blue Hole Press.
- Anslow, C., & Maurer, F. (2015). An experience report at teaching a group based agile software development project course. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education—SIGCSE '15*.
- Astels, D., Miller, G., & Novak, M. (2002). *The practical guide to extreme programming*. Prentice Hall PTR.
- Bahceci, D., & Holmgren, L. (2014). *Agile perspectives in construction projects—How to improve efficiency in the design phase* (Postgraduate). School of Architecture and the Built Environment.
- Barza, A., Cardozo, E. S., França, A. C., Neto, J. B., & Silva, F. Q. (2010). *SCRUM and productivity in software projects: A systematic literature review*. EASE.
- Battou, A. (2017). Designing an adaptive learning system based on a balanced combination of agile learner design and learner. Centered approach. *American Scientific Research Journal for Engineering, Technology, and Sciences*, 37(1), 178–186.
- Beck, K. (1999). Embracing change with extreme programming. *Computer*, 32(10), 70–77.
- Beck, K. (2000). *Extreme programming explained*. Addison-Wesley.
- Boersem, M. (2011). The Scrum Ball Point Game—Tennis, anyone?. Lean simulations. Retrieved March 18, 2018, from <http://www.leansimulations.org/2011/08/scrums-ball-point-game-tennis-anyone.html>.
- Boersem, M. (2012). Lean Paper Airplane Game Instructions. Lean simulations. Retrieved March 18, 2018, from <http://www.leansimulations.org/2012/10/lean-paper-airplane-game-instructions.html>.
- Bonwell, C. C., & Eison, J. A. (1991). Active learning; creating excitement in the classroom. *ASHE-ERIC Higher Education Report No. 1*. Washington, D.C.: The George Washington University, School of Education and Human Development.
- Bryant, S., & Romero, P. (2006). XP and pair programming practices. *PPIG Newsletter*, 17–20.
- Buildbettersoftware—Trello Guide. (2018). Using *Trello for agile software development: The complete guide*. Retrieved March 18, 2018, from <http://buildbettersoftware.com/trello-for-software-development>.
- Cervone, H. (2011). Understanding agile project management methods using Scrum. *OCLC Systems & Services: International Digital Library Perspectives*, 27(1), 18–22.
- Cohn, M. (2004). *User stories applied. For agile software development*. Boston: Addison-Wesley.
- Collabedit. (2016). Collabedit—Online text editor. Retrieved March 18, 2018, from <http://collabedit.com>.
- Corona, E., & Pani, F. E. (2013). A review of Lean-Kanban approaches in the software development. *WSEAS Transactions on Information Science and Applications*, 10(1), 1–13.
- Da Silva, I., Xambre, A. R., & Lopes, R. B. (2013). A simulation game framework for teaching Lean production. *International Journal of Industrial Engineering and Management*, 4(2), 81–86.
- Devedžić, V., & Milenković, S. (2011). Teaching agile software development: A case study. *IEEE Transactions on Education*, 54(2), 273–278.
- Dorst, K. (2011). The core of 'design thinking' and its application. *Design Studies*, 32(6), 521–532.
- D'Souza, M., & Rodrigues, P. (2015). Extreme pedagogy: An agile teaching-learning methodology for engineering education. *Indian Journal of Science and Technology*, 8(9), 828.
- Dym, C., Agogino, A., Eris, O., Frey, D., & Leifer, L. (2005). Engineering design thinking, teaching, and learning. *Journal of Engineering Education*, 94(1), 103–120.
- Erdogmus, H., Melnik, G., & Jeffries, R. (2010). Test-driven development. *Encyclopedia of Software Engineering*. Taylor & Francis.

- Famuyide, S. (2017). *Test-driven vs behaviour-driven development*. *Business analyst learnings*. Retrieved October 31, 2017, from <https://businessanalystlearnings.com/technology-matters/2014/8/13/test-driven-vs-behaviour-driven-development>.
- Fowler, M., & Highsmith, J. (2001). The agile manifesto. *Software Development*, 9(8), 28–35.
- Friesen, K. (2015). LEGO Scrum City. *Plays-In-Business*. Retrieved March 18, 2018, from <http://www.plays-in-business.com/lego-scrum-city>.
- Heikkilä, V., Paasivaara, M., & Lassenius, C. (2016). Teaching university students Kanban with a collaborative board game. In *Proceedings of the 38th International Conference on Software Engineering Companion—ICSE '16*.
- Hoda, R., Noble, J., & Marshall, S. (2008). Agile project management. In *New Zealand Computer Science Research Student Conference* (pp. 218–221).
- Hoda, R., Noble, J., & Marshall, S. (2010). Organizing self-organizing teams. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering—ICSE '10*.
- Janes, A., & Succi, G. (2014). *Lean software development in action*. Berlin, Heidelberg: Springer.
- Janzen, D., & Saiedian, H. (2005). Test-driven development concepts, taxonomy, and future direction. *Computer*, 38(9), 43–50.
- JUnit 5. (2018). Junit.org. Retrieved March 18, 2018, from <http://junit.org>.
- Jyothi, V. E., & Rao, K. N. (2012). Effective implementation of agile practices-in coordination with Lean Kanban. *International Journal on Computer Science and Engineering*, 4(1), 87.
- Kanban Pizza Game. (2013). *agile42—The agile coaching company*. Retrieved March 31, 2018, from <https://www.agile42.com/en/training/kanban-pizza-game/>.
- Keefe, K., Sheard, J., & Dick, M. (2006, January). Adopting XP practices for teaching object oriented programming. In *Proceedings of the 8th Australasian Conference on Computing Education* (Vol. 52, pp. 91–100).
- Kerievsky, J. (2001). PairDraw—Industrial Logic. Industrial Logic. Retrieved March 18, 2018, from <https://www.industriallogic.com/blog/pairdraw-2>.
- Knepper, T. (2017). Agile apex introduction video. Learn agile provided by Agilest.org. Retrieved March 18, 2018, from <http://learn.agilest.org/project-management/agile-apex-introduction-video>.
- Kollanus, S., & Isomöttönen, V. (2008). Test-driven development in education: Experiences with critical viewpoints. *ACM SIGCSE Bulletin*, 40(3), 124–127.
- Kropp, M., & Meier, A. (2013). Teaching agile software development at university level: Values, management, and craftsmanship. In *26th International Conference on Software Engineering Education and Training (CSEE&T)*.
- Kuppuswami, S., Vivekanandan, K., Ramaswamy, P., & Rodrigues, P. (2003). *The effects of individual XP practices on software development effort*. *ACM SIGSOFT Software Engineering Notes*, 28(6), 6.
- Lappalainen, V., Itkonen, J., Isomöttönen, V., & Kollanus, S. (2010). ComTest. In *Proceedings of the Fifteenth Annual Conference on Innovation and Technology in Computer Science Education—Iticse '10*.
- Lindstrom, L., & Jeffries, R. (2004). Extreme programming and agile software development methodologies. *Information Systems Management*, 21(3), 41–52.
- Lynch, T., Herold, M., Bolinger, J., Deshpande, S., Bihari, T., Ramanathan, J., & Ramnath, R. (2011). An agile boot camp: Using a LEGO-based active game to ground agile development principles. In *2011 Frontiers in Education Conference (FIE)*.
- Mahalakshmi, M., & Sundararajan, M. (2013). Traditional SDLC vs scrum methodology—A comparative study. *International Journal of Emerging Technology and Advanced Engineering*, 3(6), 192–196.
- Manifesto for Agile Software Development. (2001). *Agile manifesto*. Retrieved March 28, 2018, from <http://www.agilemanifesto.org>.
- McDowell, C., Werner, L., Bullock, H. E., & Fernald, J. (2006). Pair programming improves student retention, confidence, and program quality. *Communications of the ACM*, 49(8), 90–95.

- Michaelsen, L., & Sweet, M. (2008). The essential elements of team-based learning. *New Directions For Teaching And Learning, 2008*(116), 7–27.
- Muller, M., & Tichy, W. (2001). Case study: Extreme programming in a university environment. In *Proceedings of the 23rd International Conference on Software Engineering, ICSE*.
- Nerur, S., Mahapatra, R., & Mangalaraj, G. (2005). Challenges of migrating to agile methodologies. *Communications of the ACM, 48*(5), 72–78.
- O’heocha, C., & Conboy, K. (2010). The role of the user story agile practice in innovation. In *Lean Enterprise Software and Systems* (pp. 20–30).
- Overall Equipment Effectiveness. (2017). *Leanproduction.com*. Retrieved March 29, 2018, from <https://www.leanproduction.com/oeo.html>.
- Paasivaara, M., Heikkilä, V., Lassenius, C., & Toivola, T. (2014). Teaching students scrum using LEGO blocks. In *Companion Proceedings of the 36th International Conference on Software Engineering—ICSE Companion 2014*.
- Patton, J. (2014). *User story mapping: Discover the whole story, build the right product*. O’Reilly Media.
- Poppendieck, M., & Cusumano, M. (2012). Lean software development: A tutorial. *IEEE Software, 29*(5), 26–32.
- Poppendieck, M., & Poppendieck, T. (2007). *Implementing Lean software development: From concept to cash*. Pearson Education.
- Poppendieck, M., & Poppendieck, T. (2003). *Lean software development*. Boston: Addison-Wesley.
- Schwaber, K. (1997). SCRUM development process. *Business Object Design and Implementation, 117–134*.
- Spacco, J., & Pugh, W. (2006). Helping students appreciate test-driven development (TDD). In *Companion to the 21st ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages, and Applications—OOPSLA ‘06*.
- Svitis, C. (2013). *Lean software development: Theory validation in terms of cost reduction and quality improvement*. Unpublished bachelor’s thesis, University of Gothenburg, Chalmers University of Technology, Göteborg, Sweden.
- Swanson, B. (2013). Kanban Pizza Game at Agile Development Conference East 2013. agile42—The Agile Coaching company. Retrieved March 18, 2018, from <http://www.agile42.com/en/blog/2013/10/01/kanban-pizza-game-agile-development-conference-east-2013>.
- Unified Agile & DevOps. (2018). *VersionOne*. Retrieved March 28, 2018, from <https://www.versionone.com>.
- Waters, K. (2011). Agile Games—ball point game | 101 Ways. 101 Ways Blog. Retrieved March 18, 2018, from <https://www.101ways.com/agile-games-ball-point-game/>.
- What is a Kanban Board?. (2018). *LeanKit*. Retrieved March 29, 2018, from <https://leankit.com/learn/kanban/kanban-board>.
- Williams, L., Wiebe, E., Yang, K., Ferzli, M., & Miller, C. (2002). In support of pair programming in the introductory computer science course. *Computer Science Education, 12*(3), 197–212.
- Van Til, R. P., Tracey, M. W., Sengupta, S., & Fliedner, G. (2009). Teaching Lean with an interdisciplinary problem-solving learning approach. *International Journal of Engineering Education, 25*(1), 173.
- Yenduri, S., & Perkins, A. (2006). Impact of using test-driven development: A case study. In *Proceedings of the International Conference on Software Engineering Research and Practice & Conference on Programming Languages and Compilers, SERP 2006*.

# Using Agile Games to Invigorate Agile and Lean Software Development Learning in Classrooms



Rashina Hoda

**Abstract** A wide variety of professional certifications, trainings and dedicated academic courses are attempting to meet the ever-growing demand for software professionals competent in the knowledge and use of agile and lean software methods and practices. Agile games, embodying experiential learning, are popular in industrial contexts and are increasingly being trialed in academic settings as a feasible alternative or a complement to traditional instructional learning approaches. Most games reported, however, focus exclusively on the Scrum method and practices. This study reports on the use of four agile games for learning fundamental agile and lean concepts such as iterative and incremental delivery, collaborative estimation, pair programming and work-in-progress limits. Based on classroom observations and survey-based quantitative and qualitative data, we found that: agile games are a useful supplement for effective learning, can easily invigorate learner engagement and promote team building. Effective facilitation and debriefing sessions are imperative to the success of agile games in classrooms. Educators can easily introduce agile games by selecting from a variety of accessible online resources based on their ability to deliver desired learning outcomes and graduate attributes to invigorate learning about agile and lean software development.

**Keywords** Agile software development · Games · Engagement · Learning Teaching · Classroom

## 1 Introduction

Agile and lean software development has witnessed widespread adoption in the software industry (Deemer, Benefield, Larman, & Vodde, 2010) as well as in academia over the past two decades (Scott, Rodríguez, Soria, & Campo, 2016). Rapid response

---

R. Hoda (✉)  
SEPTA Research, Department of Electrical and Computer Engineering,  
The University of Auckland, Auckland, New Zealand  
e-mail: [r.hoda@auckland.ac.nz](mailto:r.hoda@auckland.ac.nz)

© Springer Nature Singapore Pte Ltd. 2019  
D. Parsons and K. MacCallum (eds.), *Agile and Lean Concepts for Teaching and Learning*, [https://doi.org/10.1007/978-981-13-2751-3\\_18](https://doi.org/10.1007/978-981-13-2751-3_18)

391

to frequent changes in requirements, fast delivery times, close customer collaboration, empowered self-organizing teams and adaptive management are some of the distinguishing features that have accelerated the adoption and practice of agile methods (Schwaber & Beedle, 2002; Beck, 1999; Hoda, Noble, & Marshall, 2013; Augustine, 2005). The industry's demands for software professionals trained and competent in agile and lean methods and practices are met by efforts from several professional trainers and universities offering dedicated courses on these topics (Kruchten, 2011; Wangenheim, Savi, & Borgatto, 2013). The use of capstone projects to introduce agile methods has been a popular approach in such agile courses (Mahnic, 2012; Lu & DeClue, 2011; Schroeder, Klarl, Mayer, & Kroiß, 2012; Scott et al., 2016). The theoretical concepts underlying agile and lean software development, however, still need to be imparted regardless of whether the course includes a practical component. This is typically achieved through traditional classroom instructional approaches such as lectures (Wangenheim et al., 2013; Mahnic, 2012). Making the information-laden component of the course engaging and effective is nontrivial.

Games—embodying experiential learning—are viewed as a feasible alternative or a complement to traditional instructional learning approaches (Percival, Ellington, & Race, 1993; Wangenheim & Shull, 2009). Games are seen to offer a unique opportunity to leverage high user engagement and interactivity potential combined with learning objectives (Wouters et al., 2009; Arnab et al., 2014); and provide motivation for learning (Malone, 1981) as players often display deep concentration and imbibe concepts through gameplay (Shneiderman, 2004).

Games for learning Agile software development concepts, termed in this chapter as 'Agile games', are widely prevalent in industrial training contexts and many games are easily accessible online (e.g. Heintz, 2016; Cohn, 2016; Kerievsky, 2016; Kniberg, 2016). The efficacy and ability of games to support agile learning are increasingly being trialed in academic contexts (Fernandes & Sousa, 2010; Wangenheim et al., 2013; Paasivaara, Heikkila, Lassenius, & Toivola, 2014). Games such as PlayScrum (Fernandes & Sousa, 2010), Scrumia (Wangenheim et al., 2013) and those using Lego<sup>®</sup>-bricks (Lynch et al., 2011; Paasivaara et al., 2014), designed to simulate the Scrum lifecycle, have reported evidence of successful use in academic settings. Most of these games, however, have focused on one particular agile method, Scrum, simulating the entire Scrum development cycle or a particular Scrum practice. Other concepts and practices of agile software development such as XP's popular pair programming have received relatively little attention. Similarly, Lean and Kanban concepts have remained relatively untouched in reported academic contexts.

This study involved the use four games—*Paper Planes* (Heintz, 2016), *Planning Poker* (Cohn, 2016), *Pair Draw* (Kerievsky, 2016), and the *Name Game* (Kniberg, 2016)—in a graduate-level, dedicated agile and lean software development course at the University of Auckland. These games primarily focus on concepts such as: agile software development's *iterative and incremental delivery* model; Scrum's *collective team estimation* practice; XP's *pair programming* practice; and Kanban's *work-in-progress limit* concept, respectively. They were selected from a plethora of available online Agile games based on their ability to map to the desired learning outcomes and graduate attributes of the course, as described later.

Based on classroom observations and qualitative and quantitative data collected through a specially designed survey, our main findings are that: the four Agile games effectively supplemented learning of fundamental agile and lean concepts; the games strongly invigorated classroom engagement; and promoted team building. Some of the lessons learned from the experience include: effective facilitation was found to be valuable to the efficient use of games in the classroom; and that a dedicated debriefing session was imperative for each game to help establish connections between the game experience and the underlying theoretical concepts and instigate discussions and insights.

## 2 Background and Related Works

### 2.1 Agile and Lean Software Development

Agile software development (Fowler & Highsmith, 2001) is an umbrella term that captures a set of values and principles that guide the modern, lightweight software development processes such as Scrum (Schwaber & Beedle, 2002) and eXtreme Programming (Beck, 1999). Among its core values are: people and interactions; customer collaboration; working software as a measure of progress; and embracing change. Emerging in the late 1990s, agile methods have rapidly become the default software development method of choice in the global software industry (Deemer et al., 2010).

Along similar lines and originating from the Toyota Product System (Taiichi, 1988; Womack, Jones, & Roos, 1990) is the concept of Lean software development. Manifested in concrete methods such as Kanban (Anderson, 2010), Lean propounds the values of eliminating waste, optimizing the whole, building in quality, constant learning, fast delivery, engaging everyone and continuous improvement (Poppendieck & Poppendieck, 2003). Lean values are adopted independently or often alongside agile methods to fine-tune the software development process.

There are several fundamental concepts related to agile and lean methods available in accessible formats (Deemer et al., 2010). The four key concepts that are most relevant to this research study and were used in the course are described below.

- **Iterative and incremental delivery:** is one of the defining features of agile software development where iterative refers to the short two-to-four-week work cycles within which a team develops a prioritized subset of product features; and incremental refers to the continuous integration of these features to form a complete product over the course of the project. Such an approach is in stark contrast to the traditional sequential development and delivery models which were marked by specific stages of design, development, testing and deployment (Royce, 1970). In agile methods such as Scrum, a selected set of requirements move through all the above stages and are delivery-ready by the end of every iteration (Schwaber & Beedle, 2002). The concept of such fixed time-boxed iterations, however, is not a

part of lean methods such as Kanban which focus more on continuous workflow (Wang, Conboy, & Cawley, 2012).

- **Effort estimation:** is the practice of estimating the effort involved in producing a customer requirement expressed in the form of a user story and often representing a particular product feature; including design, development, and testing. A unique aspect of effort estimation in an agile or lean context is that it not only involves project managers or technical leads but rather the whole team including developers and testers (Fowler & Highsmith, 2001). Since the user stories are typically self-assigned later in the iteration, estimation of these stories in the early planning stages needs to involve the whole team so that everyone can discuss and understand the requirements and can provide their input into the estimation based on experience (Hoda et al., 2013).
- **Pair programming:** is one of the most well-known XP practices which involves two developers working together on a single computer to develop software features together, where one writes the code, referred to as the driver, and the other is meant to provide guidance and help, referred to as the navigator (Beck, 1999). The partners swap roles frequently. Pair programming continues to be an important and popular agile practice (Williams, 2010).
- **Work-in-Progress (WIP) limit:** is the concept of placing constraints on the number of work items being executed at any given time with the aim to reduce multitasking and improve productivity. The WIP limit ensures that a pull system is followed and nothing is built before it is needed (Wang et al., 2012). Kanban's focus on limiting WIP differentiates it from most agile methods.

Following the rapid growth and popularity of agile and lean methods worldwide, many professional training courses, workshops and certificates are available (e.g. Scrum Master certification by the Scrum Alliance, Agile Certified Practitioner by the Project Management Institute). Several multinational companies are dedicated to producing tools and offering services to support agile and lean software teams (e.g. VersionOne, Rally Software, ThoughtWorks).

Similar trends have been witnessed in the education domain where courses dedicated to agile and lean software development are offered by many computer science and software engineering departments in universities around the world, e.g. University of Oxford, Carnegie Mellon, John Hopkins, Alto University, UNICEN University. While it is obvious that knowledge and experience in agile and lean methods are beneficial to the job prospects of current and future graduates, the approaches to teaching and learning of these methods vary. Using capstone projects to allow students to practice and experience agile methods has been widely reported (Mahnic, 2012; Lu and DeClue, 2011; Schroeder et al., 2012; Scott et al., 2016). While our course SOFTENG761 at the University of Auckland includes a quasi-real-world project experience component as described later, where the delivery of the fundamental agile and lean concepts prior to commencing the project in an engaging and effective manner was a challenge.



## 2.2 Learning Agile and Lean Through Games

Games have been described as goal-directed, competitive activities with agreed rules (Lindley, 2003; Wouters et al., 2009). Games whose main purpose is other than entertainment such as for learning and instruction are referred to as serious or educational games (Wouters et al., 2009). Educational games often involve simulations and role playing which are seen to embody experiential learning (Kolb, 1984).

There is a growing interest in the use of games for learning as they offer a unique opportunity to leverage high user engagement and interactivity potential combined with learning objectives (Wouters et al., 2009; Arnab et al., 2014). Furthermore, games are seen to provide motivation for learning (Malone, 1981) as players often display deep concentration and imbibe concepts through gameplay (Shneiderman, 2004).

Since agile and lean methods focus on people and interactions, collaborative games harness as well as promote these skills. Furthermore, agile methods propound rapid response and games provide hands-on opportunities to experience and promptly react to situations and contexts. It is therefore no surprise that games are popular means of teaching and learning agile concepts in professional training contexts and are being increasingly trialed in academic settings.

### 2.2.1 Industrial Offerings

In industrial training contexts, simulations, role play and games-based learning have been actively used (e.g. certification courses offered by the Scrum Alliance). A plethora of resources are freely available online to support game-driven approaches for use by industrial professionals and academics alike. The games relevant to our study include: Paper Planes (Heintz, 2016), Planning Poker (Cohn, 2016), Pair Draw (Kerievsky, 2016), and the Name Game (Kniberg, 2016). Complete descriptions and instructions for each of these games are available on their respective websites. Brief descriptions of the purpose, execution and expected outcomes of the four games are presented below to aid the understanding of the research context and results to follow.

- **Paper Planes:** is a collaborative simulation game used to introduce agile software development. It primarily focuses on the iterative and incremental delivery model and provides the opportunity to experience teamwork, customer collaboration and reflective practice. It involves multiple teams working on creating paper planes within time-boxed intervals or iterations. The facilitator explains the rules of the game and keeps time. They also act as the customer during the simulation, providing requirements and conducting acceptance tests of delivered product, for example, all planes must have blunt heads for safety and those not complying are rejected. The simulation typically includes three iterations and lasts around 2 h. Each iteration is divided into a 'plan-do-check' cycle where the plan stage involves the team planning their iteration and lasts 1 min, the do stage involves the actual production of the paper planes and lasts 2 min, and the check stage involves

reflection as a team and lasts 1 min. Teams are typically able to understand the importance of planning and reflection as their productivity expectations change from being over-inflated to realistic over the course of the simulation. Different versions of this popular game are available online along with details of materials required, play instructions, and game resources (Heintz, 2016).

- **Planning Poker:** is a collaborative gamified estimation technique used for effort estimation in Scrum (Cohn, 2016). Instead of estimating real user stories, in a simulated, training context, the learner teams are asked to estimate the sizes of various items, such as animals on a scale of one to ten, where one is the smallest and ten represents the largest. A photo of an animal is presented and individuals are asked to think of a size rating in their mind without sharing or discussing with others. On the count of three, all learners are asked to show their rating by raising equivalent number of fingers (or cards printed with numbers). The facilitator then goes up to each team and singles out the individuals with the maximum difference in estimations, e.g. one and four, for the same animal and asks them to discuss their rationale. The idea is to allow team members to hear and understand other's perspectives and potentially refine their estimation based on the new information. The discussions continue until the gap is closed or lessened. This is repeated for other animals. As new information becomes available to teams, they may decide to recalibrate their previous estimations. The simulation enables individuals to appreciate multiple perspectives gained during collaborative estimation and learn to think and estimate as a team as opposed to individuals. The game usually takes around an hour including time for debriefing. Various rating scales include sequential numbers, the Fibonacci series or T-shirt sizes (extra small, small, medium, large, extra large) and others.
- **Pair Draw:** (Kerievsky, 2016) is a game which simulates XP's pair programming practice. First, the learners are asked to individual draw a face on a sheet of paper. The drawings can be as simple or as detailed as they prefer. Then, the facilitator asks the learners to form pairs (having an even number of learners helps). The exercise is repeated but this time, the pairs need to coordinate with each other to draw a single face collaboratively. Individuals within a pair are asked to use different coloured pens so it is easy to tell which bits were drawn by whom. Pairs can share their experiences and drawings with the wider group. The simulation is followed by a debrief session where the facilitator highlights the similarities between the simulation and the actual pair programming practice. Other questions are also posed to the group for discussion, such as what was the difference in drawing alone and drawing as a pair? What were the advantages and disadvantages of each? These allow the learners to form some expectations of what actual pair programming may entail.
- **The Name Game:** (Kniberg, 2016) is a simulation game that helps illustrate how multitasking can reduce efficiency and productivity. It is used to enforce the concept of applying work-in-progress (WIP) limits in Kanban. It involves team members playing the role of customers providing their names while one of the team members plays the developer who writes the names down. Two iterations of the game are played. In the first iteration, each customer provides a single letter of

their name at a time to the developer who writes it down on the customer's piece of paper and then attends to the next customer. This continues until all customers are served (i.e. their complete names are written). This scenario represents multitasking as the developer simultaneously works on all customer requests. The second iteration involves the same task but the approach is different. Here, the developer attends to one customer at a time, writing their full name down before moving on to the next customer. In other words, a WIP limit of one customer request at a time is applied. It is usually noticed that the second scenario (with WIP limit in place) yields better productivity and generally better experiences for the developer and customers.

### 2.2.2 Academic Adoption

In the academic context, the use of games and simulations for imparting software engineering concepts has been reported (Drappa & Ludewig, 2000; Baker, Navarro, & van der Hoek, 2005; Carrington, Baker, & van der Hoek, 2005). The use of games for learning agile and lean concepts has also been trialed in various contexts. Fernandes & Sousa (2010), Lynch et al. (2011), Wangenheim et al. (2013), and Paasivaara et al. (2014) form a growing body of evidence in support of using games for teaching and learning agile concepts.

PlayScrum—a card game devised for use in the university context (Fernandes & Sousa, 2010). The focus of the game was to provide players with the experience of the Scrum method and the role of the Scrum Master. Feedback from 13 Masters level students who trialed the game found it be to visual, simple, useful and enjoyable. The authors noted that the game was best used to complement the teaching of the theoretical concepts in class.

Wangenheim et al. (2013) presented and used Scrumia—a physical paper-based game, as a low-budget approach to complement more the traditional classroom instruction of Scrum. It involved simulating a scrum development cycle in teams of six people. The study provided evidence that the game provided a positive effect on learning Scrum. Like Fernandes & Sousa (2010), Wangenheim et al. (2013) also propound that such games are effective in complementing theoretical lectures.

The use of Lego®-based games has also been explored in academic settings (Lynch et al., 2011; Paasivaara et al., 2014). Lego® bricks were used to build physical products in an agile boot camp setting involving learning across three iterations (Lynch et al., 2011). Students showed a preference for the agile boot camp over lecture-based teaching. Paasivaara et al. (2014) used a Lego®-based simulation game in a Masters level course at Alto University. Surveys of learner experiences showed high levels of satisfaction. While Lynch et al. (2011) found no difference in the recall of concepts introduced through games and those in lectures, Paasivaara et al. (2014) reported gains in learning and insight as a result of gameplay.

As explained above, numerous low-budget, paper-based games originating from industry are easily accessible online while some have originated from the academic domain (Fernandes & Sousa, 2010; Wangenheim et al., 2013). Most of these games

have focused on one particular agile method—Scrum—and aimed to simulate the entire Scrum development cycle or a particular Scrum practice. Other concepts and practices of agile software development such as XP’s popular pair programming have received relatively little attention. Similarly, lean and Kanban concepts have remained relatively untouched in academic contexts.

Our research study reports on the use of four Agile games—Paper Planes, Planning Poker, Pair Draw and the Name Game—for learning agile software development’s iterative and incremental delivery, Scrum’s collective team estimations, XP’s pair programming and Kanban’s WIP limits concept respectively, in an academic context and provides evidence to support the games-driven approach for agile and lean learning.

### 3 Research Context and Design

#### 3.1 The Course Context

SOFTENG761—*Agile and Lean Software Development* is a course offered to final year bachelor of engineering (BE) and masters of engineering studies (ME studies) learners specializing in software engineering in the Electrical and Computer Engineering department at the University of Auckland. Like most courses at the university, this is a 12-week course with a 2-week break typically separating two parts of 6 weeks each. The ratio of BE final year and the ME studies learners tends to vary between three to two (3:2) and equal (1:1). The course was first designed and launched by the author who has continued to run the course in the capacity of the course director and lecturer. It has since witnessed consistent interest and growth, reflecting the growth of the software engineering program. Learners enrolling into the course are expected to possess advanced programming skills in popular languages such as Java, and have prior academic or industrial experience of team-based software projects. Their advanced skills and experience are expected to help them select from a variety of projects to implement, requiring a multitude of latest technologies and tools.

The main purpose of the course is to teach: the fundamental concepts of agile and lean software development such as iterative and incremental software development, self-organizing teamwork, customer collaboration and project management; and the core practices from the Scrum, XP, and Kanban methods such as release and iteration planning, pair programming and team estimations. The teaching approach for SOFT-ENG761 is based on a three-tier learning approach comprising of three progressive parts focusing on:

- **Part I—Theory:** provides learners with the fundamental knowledge about agile and lean software development over the first 3 weeks, typically imparted in the form of direct lecture-based instruction using PowerPoint slides, supplemented by videos and classroom discussions. Theory is tested at the end of week three in a test worth 25%.

- **Part II—Practice:** provides learners with the opportunity of hands-on practice in agile and lean software development. Learners are required to self-form teams and select from the project options offered by local software companies. Students typically form teams of seven to eight students each. The projects cover a 6–7 week period following weekly iterations and hands-on practice of several Scrum, XP and Kanban practices. This makes up 50% of the assessment. Further details of the projects and the industry collaboration aspects of this course are beyond the scope of this chapter.
- **Part III—Research:** requires learners to search and review current research literature on various agile and lean-related topics with a focus on critically analysing these concepts in light of literature and their own project experiences and presenting them as an individual research essay worth 25%.

### 3.2 *The Challenge*

Delivering fundamental knowledge on any subject requires the definition and description of several new terminologies, concepts and ideas. In the earliest offering of the course, this was achieved through the instructor talking through PowerPoint slides to a classroom full of learners with occasional questions and answers. In a bid to make things more interactive, learners were presented with scenarios from software development contexts and encouraged to discuss and debate possible outcomes relevant to agile practices. The main challenges associated with this approach were: level of learner engagement remained average as only some learners participated in the question–answer sessions or discussions around contexts while a majority of the class remained in passive recipient mode. As such it was unclear to what extent the class was engaged and whether the learners effectively comprehended the concepts being covered.

### 3.3 *Introducing Games*

Games were introduced into the course to improve the level of engagement and learning in the theory part of the course. A set of four different Agile game (games for learning Agile software development) were identified and added to the instructional materials and design. These were selected based on their ability to align to the course’s desired learning outcomes and set of graduate attributes listed by the Washington Accord (Accord, 2013), an international agreement between bodies responsible for the accreditation of tertiary level engineering qualifications. Employing a set of graduate attributes defined by the Accord, such as *WA09 individual and team work*, provides a valuable framework for achieving constructive alignment with learning outcomes during design of engineering courses. A mapping between the games, learning outcomes and graduate attributes is presented in Table 1.

**Table 1** Mapping of collaborative games deployed, underlying agile and lean concepts, course learning outcomes, and graduate attributes defined by the Washington Accord (Accord, 2013)

Agile games	Underlying agile and lean concepts	Learning outcomes	Most relevant graduate attributes <sup>a</sup>
Paper Planes	Iterative and incremental software delivery	LO 1. Plan for project release and iterative delivery LO2. Develop, test, and deliver software in an iterative and incremental fashion	WA09, WA10, WK04, WK06, WA11
Planning Poker	Collective team estimations	LO3. Estimate user stories	WA09, WA10, WK04, WK05, WK06
Pair Draw	Pair programming	LO4. Work effectively in a self-organizing team environment	WA09, WA10, WK04, WK06
Name Game	Work-in-progress (WIP) limit	LO5: Design strategies to overcome common challenges of Agile and Lean practices	WA09, WA10, WK04, WK06, WP06

<sup>a</sup>WA09: individual and team work; WA10: communication; WK04: specialist knowledge; WK05: engineering design; WK06: engineering practice; WA11: project management and finance; WP06: conflicting stakeholder requirements

As the games were played prior to the projects commencing, learners from the same project teams were asked to play together. All games were played as project teams, except for Pair Draw, where learners were asked to form pairs within teams. Brief descriptions of these games have been provided earlier in (Heintz, 2016; Cohn, 2016; Kerievsky, 2016; Kniberg, 2016) and complete details are available on their respective websites as referenced.

The games were played in classroom sessions following a brief introduction to the relevant concepts via PowerPoint slides and in-class discussions. Two-hour sessions were found to be necessary and sufficient for including some instructional material and at least one game. However, the Paper Planes game took up the entire 2 h as it involved several steps and iterations. The games were facilitated by the lecturer/author with the help of a teaching assistant in charge of time keeping. The lecturer/author is an agile researcher and a certified Scrum Master with wide experience of teaching agile and lean concepts and practices. The lecturer moved around the entire classroom, encouraging learners through questions about their approach and progress. All learners were seen to actively engage in the games. Further details of classroom experiences are discussed and shown later in the Findings section.

### 3.4 Data Collection and Analysis

Human ethics approval was gained prior to collecting data from the learners. The game sessions were observed by the lecturer and the teaching assistant. Notes were taken to record these observations. Quantitative and qualitative data were collected from learners about their experience of playing the games in the course through a specially designed games survey. The games survey was created and hosted on Google Forms and was conducted soon after the last game was played in class and the theory part of the course was over, in week four. The form was set up to require user authentication using the university credentials to ensure only learners from the course participated, however, the details of the participants were not stored to allow for anonymous and more candid responses. The form was further set to accept only one response per person. A link to the survey was emailed out to the class and participation was completely voluntary. Twenty-three of the total 51 learners (45%) responded to the survey.

The games survey included a set of questions about games for learning in general and about the experience of playing specific games in class (see full list in Appendix). The questions were of three types: multiple choice questions (MCQs), ratings using Likert scale (1–5) type questions, and open-ended questions, for example:

- **[MCQ]** The number of games used in SOFTENG761 was: Too few/Just right/Too many.
- **[Rate on scale of 1–5 where 1 is strongly disagree and 5 is strongly agree]**
- It was more effective to learn through collaborative games than learning on my own in class (e.g. listening to lectures).
- **[Open-ended questions]**
- What did you like about the games in class?
- What didn't you like about the games in class? Please share ideas for improvements.

The observation notes were synthesized to draw out insights into the games' execution and outcomes. The games survey results were analysed and presented in graphical format by the Google Forms tools. The open-ended answers were manually analysed to identify the most significant concerns of the learners following thematic analysis (Braun & Clarke, 2006). The results of the surveys and qualitative findings are presented in the next section.

## 4 Findings

Observations of the games played in class and the responses to the games survey showed a strong support for games-driven learning in classrooms in general and for learning about agile and lean software development concepts. We first present the observations made in the classroom, followed by the survey results pertaining to specific games played in the classroom and then report on learners' perceptions of the use of games for learning in general.

## 4.1 *Classroom Observations*

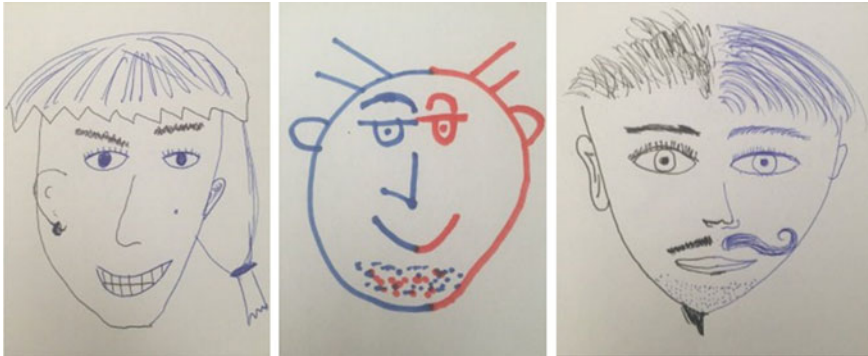
All learners were seen to engage in the Paper Planes game. High levels of teamwork and competition between teams were observed. In the first iteration, all teams overestimated their productivity, most left testing to the end, running out of time. When the ‘customer’ (lecturer/author) inspected the deliverables (planes) at the end of the iteration, several planes were rejected because they did not meet the acceptance criteria. This led to teams paying closer attention to the customer’s acceptance criteria in future iterations. In the ‘check’ (reflection) stage, teams could discuss what was working well for them and what improvements they needed to make; both to the design of the planes (technical improvements) and their approach to production and testing (process improvements). Most teams adjusted their estimates to more realistic numbers in the second iteration and commenced testing early in the ‘do’ stage, some ascribing a dedicated tester. The productivity rose incrementally until it peaked in the third iteration where teams had now begun to fine-tune their designs and process. The final debriefing session allowed learners to share their thoughts on the experience. Learners could appreciate the value of explicit ‘plan’ and ‘check’ stages in addition to the usual ‘do’ stage. This was particularly useful for the final year undergraduate students who are often seen to place more value on software coding or development and underestimate the importance of design, planning and reflection. They also observed the value of iterative delivery to deliver constant value to the customer and receive early feedback on design which could be incorporated into future iterations.

In the Name Game, one team member played the developer and the remaining members played the customers. Customers were seen waiting for their turn to interact with the developer. When asked about this in the debriefing session, most ‘customers’ agreed that they preferred longer waiting in some cases (e.g. the last few customers in queue) to receive the full, undivided attention of the developer to their request than to be served simultaneously alongside other customers. One of the customers didn’t find any observable difference in their experience as they were the last to be served in both scenarios. In the real world, however, with WIP limits in place, customers wouldn’t necessarily need to wait idly while the developer is serving others and can in fact attend to their own work until their request is ready for action.

It was seen that the whole class was engaged in the Planning Poker game, placing their hands up to represent their individual estimations by number of fingers and looking around to see with interest what their teammates and other peers estimated. The facilitator/author interacted with some teams to demonstrate the process of discussing and resolving estimation differences. It is important that the facilitator explains the importance of such discussions and clarifies that the intent is to enable outliers to voice their multiple perspectives rather than to pick on them.

Figure 1 shows some of the faces drawn as pairs during the Pair Draw game. The three drawings were drawn by three different learner pairs. Within each pair, individuals took turns to draw a single face feature using different coloured pens. The first drawing shows an approach where the two individuals drew different complete





**Fig. 1** Face drawings created in pairs during the Pair Draw game; each of the three drawings was created by a pair each using two different coloured pens (e.g. blue and black in the first drawing.)

parts of the face (e.g. both eyebrows drawn in black by one and both eyes drawn in blue by another).

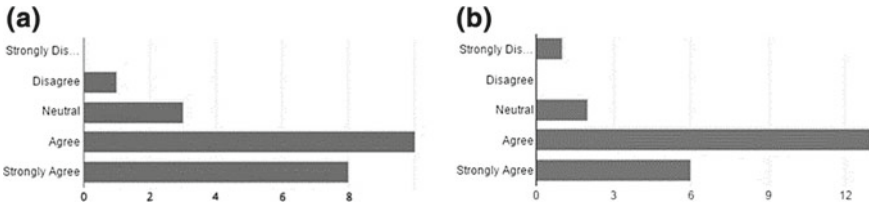
The second and third drawings show an approach where the individuals attempted to mirror each other's actions on either side of the face. It was observed that often individuals tended to mirror their pair, e.g. if an individual drew the right eye, the other drew the left eye and so forth. Some were more creative in their efforts than others, but overall most pairs attempted to create a coherent face together. As the facilitator, it is important to reassure learners that the aim of the game is not to assess their drawing skills rather to allow them to experience working on a single task as pairs as some learners can be inhibited otherwise.

## 4.2 Games Survey Results: Specific Games

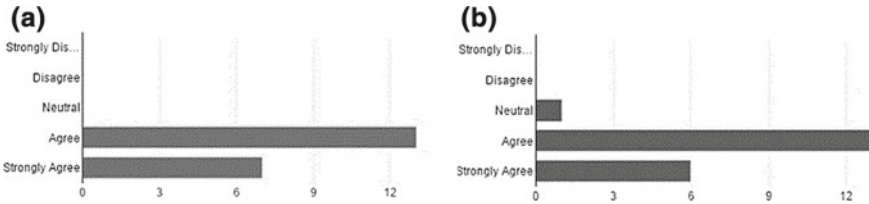
When asked about the ability of the game Paper Planes to teach the underlying agile concept of iterative delivery in the games survey, most learners (n = 18 of 23) agreed or strongly agreed while only one learner disagreed.

We further probed about the game's effectiveness despite not involving any actual software delivery. Most of the respondents still agreed. Similar questions were asked for the remaining games.

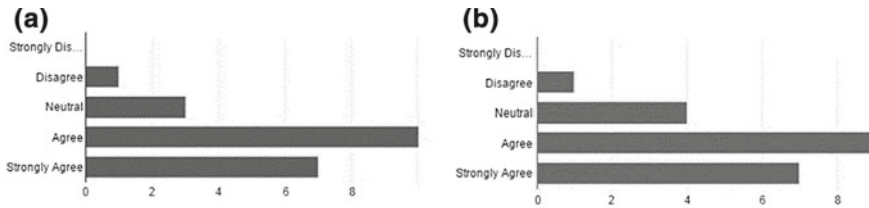
Figures 2a, 3a, and 4a show the responses for the games Planning Poker, Pair Draw, and Name Game to teach estimation, pair programming, and work-in-progress limits, respectively. Figures 2b, 3b, and 4b show the same when emphasizing the non-software nature of the games.



**Fig. 2** **a** Effectiveness of Planning Poker game to teach effort estimation. **b** Effectiveness of Planning Poker game to teach about effort estimation, despite not involving actual user stories



**Fig. 3** **a** Effectiveness of Pair Draw game to teach about pair programming. **b** Effectiveness of Pair Draw game to teach about Pair Programming, despite not involving actual programming

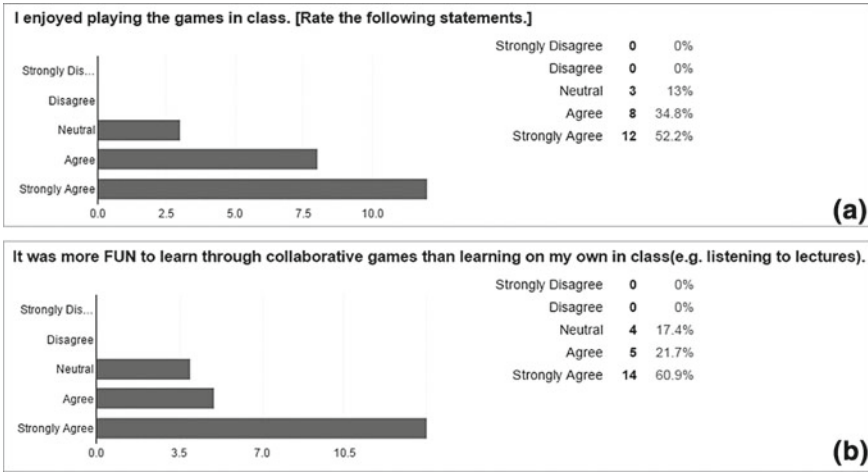


**Fig. 4** **a** Effectiveness of Name Game to teach about Work-in-Progress Limits. **b** Effectiveness of Name Game to teach about Work-in-Progress Limits, despite not involving software work

### 4.3 Games Survey Results: General Questions

In introducing games into the curriculum, we wanted to include an optimum number of games in classroom sessions such that the learners felt neither over- nor underwhelmed. Most respondents said that the number was ‘just right’, while 3 learners said there were ‘too few’. None of them thought that the number of games was ‘too many’ suggesting that we had managed to achieve a reasonable balance with four games in a 3-week learning period.

We posed a series of questions aimed at assessing the learners’ opinions of games for learning as a means to improve fun and enjoyment leading to better engagement in classroom learning. An overwhelming majority of learners enjoyed playing the collaborative games in class as seen from the results in Fig. 5a. Most of the learners (n= 19 of 23) found collaborative games to be more fun than individual or solo learning in a classroom setting as depicted in Fig. 5b.



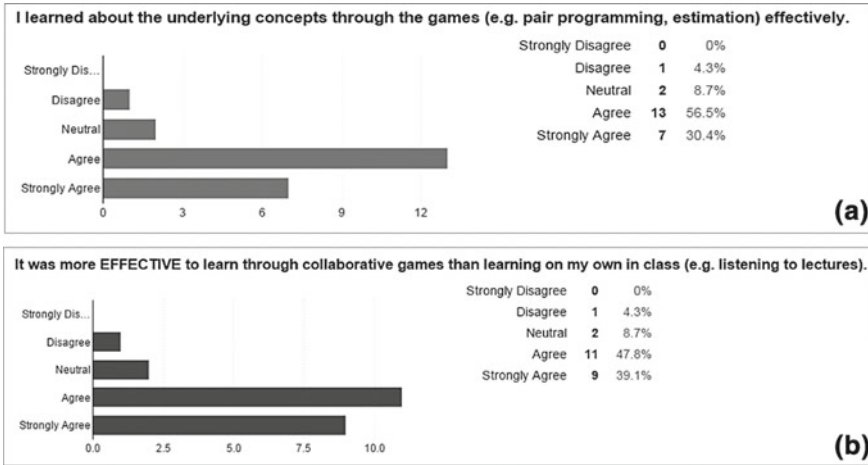
**Fig. 5** **a** Perceived enjoyment in playing games in the class. **b** Comparing collaborative games to individual learning in classroom based on perceived levels of fun and engagement

While the ability of games to improve engagement is almost intuitive and well established (Wangenheim et al., 2013; Scott et al., 2016), evidence to support their learning effectiveness can be elusive. We posed a series of questions to assess the learners’ opinions of the games as effective classroom learning mechanisms. Most of the learners (n = 20 of 23) agreed that they learned about the underlying agile and lean concepts effectively through the games played in class as shown in Fig. 6a. Furthermore, when comparing individual learning to collaborative games-based learning in classrooms, most (n = 20) also agreed that the latter was more effective for their learning than the former, as depicted in Fig. 6b. Only one learner disagreed in each case.

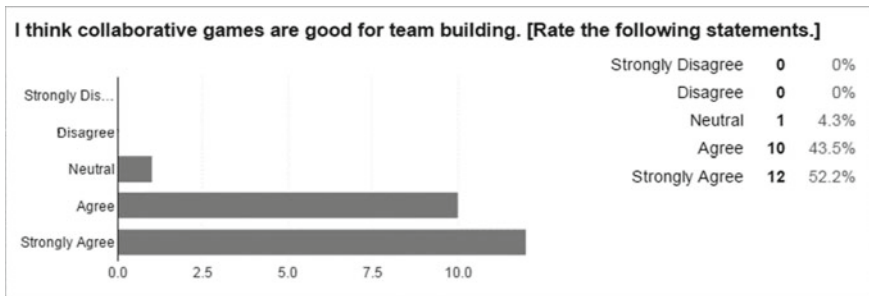
We also explored the learners’ opinions on the ability of collaborative classroom-based games to assist with team building. An overwhelming majority of learners (n = 22 of 23) agreed that collaborative games were good for team building, as shown in Fig. 7.

Finally, we asked a set of questions to gauge the learners’ perceptions of games for learning in general. Many learners (n = 16) agreed that teaching through games was both useful and necessary, as depicted in Fig. 8a. Some learners (n = 6) found games to be useful for teaching, but not necessary and one learner responded by saying they neither found games useful nor necessary for teaching.

When asked about the appropriate role of games in delivery of content, most learners (n = 19) selected games to supplement traditional delivery of content while a small percentage (n = 4) said that games can replace traditional delivery, as shown in Fig. 8a and 8b, respectively. As seen in Fig. 8c, respondents (n = 18) strongly/agreed that games should be used more often in classroom teaching in general, clearly implying a strong support for games-based learning in modern classrooms.



**Fig. 6** **a** Perceived learning effectiveness of games. **b** Comparing collaborative games to individual learning in classroom based on perceived learning effectiveness



**Fig. 7** The ability of collaborative games to assist with team building

### 4.4 Games Survey Results: Open-Ended Questions

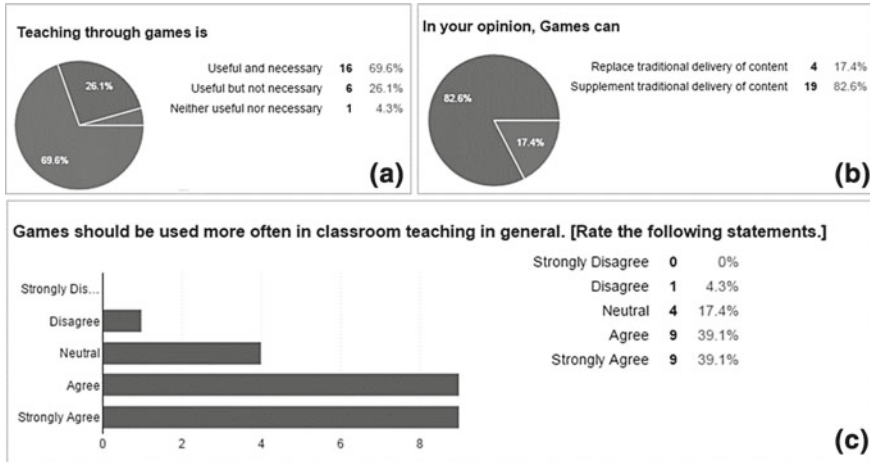
In addition to structured questions, we also included some open-ended questions. When asked about what they liked about the games in class. The responses to these questions were analysed to group concept aspects together into themes. Here the main themes, italicized below, are presented along with relevant sample quotes.

Learners said that the *interactive nature of the games improved engagement* and saw their usage as a *welcome alternative to traditional lecturing*.

Helped to break up two-hour lectures keeping the class fresh and engaged.

It helped teach the concepts in Agile in a more fun and interactive way. Rather than listening to the lecturer go over the concepts, it was more fun to play a game and then understand how it relates to Agile.

It makes the learning more engaging.



**Fig. 8** a Teaching potential and role of games. b The role of games in the delivery of content. c The use of games in classrooms in general

It is fun. Classes filled by boring speeches make me sleepy. Games in class are really good for me to focus on the content easily. I can learn knowledge with pleasure.

It was also evident from the comments that some learners found games to be *an effective way of learning key agile and lean concepts*, for example:

Simple to do, which made it easier to understand the underlying concepts related the game.  
Created a fun environment for pupils, understanding key concepts at the same time.

Many learners pointed to the collaborative games’ ability to assist with *team building*, further strengthening similar observations from the classroom and the ratings questions:

All the team work together and we can get more closer.  
It helped us work as a team. Our main aim was to succeed as a team rather than an individual.

When asked about what they didn’t like about the games in class and ideas for improvement, two respondents noted that they didn’t find Pair Draw particularly relevant to real-world pair programming. The same observation resonated in the results of the rating questions, where Pair Draw was rated least among the four games with regards to its ability to explain underlying agile and lean concepts. This may be because the Pair Draw exercise while presenting an opportunity to practice collaborative working, did not represent some of the other aspects of pair programming well, such as the ability to question each other, clarify uncertainties, make and correct mistakes, thereby carrying lesser real-world relevance than the other games.

Another improvement suggested by a few learners was *keeping the game instructions visible*. While instruction slides were provided for the Paper Planes and the

Name Game games, other games such as Pair Draw and Planning Poker were described verbally.

A bit hard to know what exactly we had to. Maybe a slide or printout with the instructions instead of being mostly verbal.

The rules of the game are sometime confused. It is better to put the rules on slides and we can see it clearly.

We have since then added written instructions for these games in the classroom to cater to all types of learners, not just those who follow verbally. We also asked the learners about other agile and lean concepts they would have liked to learn through games. Some suggestions included: daily scrum, writing user stories, the pull system from Kanban, the role of the scrum master, and use of product and sprint backlog, and games that helped compare scrum, XP and Kanban approaches. Of these, we have recently included the Kanban Pizza Game (Agile42, 2015) for simulating the pull system and WIP limit in Kanban.

## 5 Discussion

Classroom observations and the qualitative and quantitative results of the games survey showed strong evidence in support of collaborative games in the classroom. The key findings, lessons learned and related implications for practice resulting from this study are discussed below and summarized in Table 2.

Based on the experience of introducing games in SOFTENG761, it can be said that games are a useful way to supplement the learning of agile and lean software development concepts and practices. The use of traditional lecturing using PowerPoint in-class discussions and question–answers was seen to provide the necessary albeit brief background information before learners could engage in gameplay. With some background information, the learners were familiarized with the concepts underlying the game or exercise so they could draw better connections between the game experience and the learning content.

While most of the learners preferred games over traditional forms of delivery, most also subscribed to the view that games can supplement traditional delivery rather than replace it. Similar conclusions were drawn by Fernandes and Sousa (2010) and Wangenheim et al. (2013) who reported that their physical games PlayScrum and Scrumia respectively were best used in combination with traditional classroom instruction of the basic theoretical concepts. As such, trainers and educators, especially those in the academic domains, can consider selecting from a plethora of Agile games available online. While not all may be suitable, games can be assessed and selected based on how well their various components map to the desired learning outcomes and graduate attributes (Table 1).

Considered from a theoretical perspective, learning via games seems to embody reflection-in-action from Schön's theory of reflective practice (Schön, 1983) where

**Table 2** Key findings, lessons learned and implications for practice

Main findings and lessons learned	Implications for practice
The games effectively supplemented learning	Educators and trainers should consider introducing collaborative games to supplement the content shared in traditional lecture-based delivery. Using them in isolation is not advised as typically some background information is required before learners can indulge in the games and gain insights from them. Games easily available online can be selected based on their ability to map to desired learning outcomes and graduate attributes
The games improved classroom engagement	Collaborative games can be used to easily invigorate the engagement of learners in the content and in the classroom in a fun, interactive way. Team-based games promote interpersonal interaction, attract interests from a wide variety of learners, and tend to engage all learners instead of a selective few
The games promoted team building	Educators and trainers can use collaborative games to promote team building prior to commencing team-based agile projects
Effective facilitation was vital	Facilitators should not only explain the game rules and keep time but also actively encourage participation and discussion throughout the game. New facilitators should playtest with colleagues or assistants beforehand
Debriefing was imperative	Enough time should be allocated to debriefing each game. Learners should be encouraged to share experiences, insights, and questions. Most games include a list of questions to drive the debriefing session

reflection occurs during a given action (in this case, the game), as opposed to post the action occurring.

There was ample evidence of improved engagement from the classroom observations and the survey results. This is in line with similar assertions by Shneiderman (2004) and Paasivaara et al. (2014) who reported gains in learning and insight.

An additional finding from our study was that collaborative classroom games can be used effectively for team building. While this was not the primary purpose of the games, they served as a good way to break the ice within the newly formed teams where members were not familiar with each other and promote cohesion in others. This can be especially useful in courses which include a practical project-based component.

Effective facilitation was found to be useful. The facilitators not only explained the rules of the game and kept time for the different activities but also moved around the classroom to actively encourage participation and discuss how the different teams

were executing the various game activities. It would be best for new facilitators to familiarize themselves with the game and playtest it with colleagues or teaching assistants prior to implementing in the classroom.

A dedicated debriefing session provided the opportunity to verbalize and establish the connections between the game experience and the underlying theoretical concepts. It allowed not only the learners to reflect on their game experience but also enabled the facilitator to discover new aspects about the game shared by learners. The importance of such post-game debriefing—embodying reflection-on-action (Schön, 1983) where reflection occurs after the action (in this case, the game) has finished—has been previously emphasized by Crookall (2010).

### ***5.1 Limitations and Future Work***

While the facilitator and learners could play all the games within the given lecture-style classroom setup, it was observed that a different, more open-plan room setup would potentially be better for the game execution. For example, in the paper planes games, it would have been better to have an open area to test the airplanes. Similarly, for the name game and planning poker, a round table seating for each team would have helped them face each other more easily and potentially improve collaboration. In the future, we aim to request for more open-plan rooms for our course.

The Pair Draw game received the least amount of support among all the four games. Many students had no prior concept or experience of pair programming and linking the Pair Draw to pair programming may have been a wide leap for some. However, it is also a reflection on the relative low relevance of this game, Pair Draw, to the real Agile practice it was meant to represent, pair programming, as discussed earlier. We have since then dropped this game, and introduced the Kanban Pizza Game as discussed earlier.

Finally, while the evaluation of the games through student feedback, questionnaires and surveys is not the most robust or only method, it is a common method used in other studies on use of games for agile learning (Fernandes and Sousa, 2010; Lynch, 2011; Wangenheim et al., 2013). The survey instrument used in this study was developed from scratch and some of the questions could have been better phrased for clarity and validity. For example, in hindsight, the question on the ‘effectiveness’ of the games for learning compared to learning on one’s own could have captured responses comparing active and passive learning, rather than game and non-game-based learning. To better address such issues in future studies, we plan to find, and if needed extend, validated and well-established questionnaires to assess learner satisfaction and explore the possibility of pre/post testing.



## 6 Conclusion

Games are widely used in industrial agile software development certification and training workshops. There is an increasing trend of trialling games for learning agile software development in academic contexts, in particular, the Scrum method and its practices. We introduced four paper-based collaborative games in a dedicated agile and lean software development course to help learners understand and experience concepts such as iterative and incremental delivery, team-based effort estimation, pair programming, and work-in-progress limits. Through classroom observations and survey-based quantitative and qualitative data, we found that collaborative games were preferred by most learners over traditional ways of individual learning in class, both in terms of their potential for fun and learning effectiveness. These collaborative games were found to supplement learning; helped invigorate learner engagement; and promoted team building. While new games originating from academic contexts are welcomed, educators can select from a variety of existing agile games easily accessible with resources online to introduce games into their classrooms based on their ability to deliver desired learning outcomes and graduate attributes. The successful use of the games, however, is dependent on effective facilitation by knowledgeable and experienced educators and on dedicated debriefing sessions that relate the game experience with the underlying theoretical concepts. Other types of games in academic contexts and industry-based professional agile certifications, trainings and workshops can be studied in the future to gauge the impact of Agile games on engagement and learning effectiveness.

**Acknowledgements** Sincere gratitude is due to the students of SOFTENG761 Agile and Lean Software Development course at the Department of Electrical and Computer Engineering at The University of Auckland, New Zealand.

## Appendix: Survey Questions

### *Part 1: General Questions*

- Number of games used in SOFTENG761 were: Too few/Just right/Too many
- Teaching through games is: Useful and necessary/Useful but not necessary/Neither useful nor necessary
- In your opinion, games can: Replace traditional delivery of content/Supplement traditional delivery of content
- Rate the following statements (Strongly Disagree/Disagree/Neutral/Agree/Strongly Agree)
  - I enjoyed playing the games in class.
  - I learned about the underlying concepts through the games (e.g. pair programming, estimation) effectively.

- Learning through playing games teaches concepts in a way that traditional lectures cannot.
  - It was awkward to play games in class.
  - Games cannot teach concepts effectively.
  - It was more EFFECTIVE to learn through collaborative games than learning on my own in class (e.g. listening to lectures).
  - It was more FUN to learn through collaborative games than learning on my own in class (e.g. listening to lectures).
  - I think collaborative games are good for team building. o Games should be used more often in classroom teaching in general.
- Open-Ended Questions
    - What did you like about the games in class?
    - What didn't you like about the games in class? Please share any ideas for improvements.
    - What other Agile and Lean concepts would you have liked to learn through a game-based learning approach?
    - What other kinds of learning approaches or content delivery styles (other than games) would you like us to try in SOFTENG761?
    - Would you like to play such games online to learn other Agile and Lean concepts?

***Part 2: Game-Specific Questions (Repeated for Each Game, Shown for Pair Draw Here.)***

- Did you play PAIR DRAW in the class? Yes/No
- Rate the following statements (Strongly Disagree/Disagree/Neutral/Agree/Strongly Agree)
  - I found the game effective in teaching me the concept of PAIR PROGRAMMING.
  - I am more likely to attempt PAIR PROGRAMMING as a result of playing PAIRDRAW.
  - PAIRDRAW helped me understand the different aspects of PAIR PROGRAMMING even though it did not involve actual programming.
  - I expect to have similar experiences with PAIR PROGRAMMING on the actual project as I did with the PAIRDRAW simulation in class.

## References

- Accord, W. (2013). Graduate attributes and professional competencies. Version, 3, 21. Retrieved May 8, 2018, from <http://www.ieagreements.org/accords/washington/>.
- Agile42. (2015). *Kanban Pizza Game*. Retrieved May 9, 2018, from <https://www.agile42.com/en/raining/kanban-pizza-game/>.
- Anderson, D. (2010). *Kanban: Successful evolutionary change for your technology business*. Blue Hole Press.
- Arnab, S., Ger, P. M., Lim, T., Lamerias, P., Hendrix, M., Kiili, K., ... & Dahlbom, A. (2014, July). A conceptual model towards the scaffolding of learning experience. In *International conference on games and learning alliance* (pp. 83–96). Cham: Springer.
- Augustine, S. (2005). *Managing agile projects*. Prentice Hall PTR.
- Baker, A., Navarro, E., & van der Hoek, A. (2005). An experimental card game for teaching software engineering. *Journal of Systems and Software*, 75(1–2), 3–16. <https://doi.org/10.1016/j.jss.2004.02.033>.
- Beck, K. (1999). *Extreme programming explained: Embrace change* (1st ed.). Addison-Wesley Professional.
- Braun, V., & Clarke, V. (2006). Using thematic analysis in psychology. *Qualitative Research in Psychology*, 3(2), 77–101. ISSN 1478-0887.
- Carrington, D., Baker, A., & van der Hoek, A. (2005). It's all in the game: Teaching software process concepts. In *Proceedings of the 35th Frontiers in Education Conference*.
- Cohn, M. (2016). Planning Poker Game. Retrieved May 9, 2018, from <https://www.planningpoker.com/>.
- Crookall, D. (2010). Serious games, debriefing, and simulation/gaming as a discipline. *Simulation & Gaming*, 41(6), 898–920.
- Deemer, P., Benefield, G., Larman, C., & Vodde, B. (2010). The scrum primer. Scrum Primer is an indepth introduction to the theory and practice of Scrum, albeit primarily from a software development perspective. Retrieved from <http://assets.scrumtraininginstitute.com/downloads/1/scruprimer121.pdf>, 1285931497, 15.
- Drappa, A., & Ludewig, J. (2000, June). Simulation in software engineering training. In *Proceedings of the 22nd international conference on software engineering* (pp. 199–208). ACM.
- Fernandes, J. M., & Sousa, S. M. (2010, March). Playscrum—a card game to learn the scrum agile method. In *Games and virtual worlds for serious applications (VS-GAMES)*, 2010 second international conference on (pp. 52–59). IEEE.ö
- Fowler, M., & Highsmith, J. (2001). The agile manifesto. *Software Development*, 9(8), 28–35.
- Hoda, R., Noble, J., & Marshall, S. (2013). Self-organizing roles on agile software development teams. *IEEE Transactions on Software Engineering*, 39(3), 422–444 (IEEE).
- Heintz, J. (2016). Agile Airplane Game, Gist Labs. Retrieved May 9, 2018, from <http://gistlabs.com/2011/06/agile-airplane-game/>.
- Kerievsky, J. (2016). PairDraw, Industrial Logic. Retrieved May 9, 2018, from <https://www.industriallogic.com/blog/pairdraw-2/>.
- Kniberg, H. (2016). Multitasking Name Game, Crisp. Retrieved May 9, 2018, from <https://www.crisp.se/gratis-material-ochguider/multitasking-name-game>.
- Kolb, D. A. (1984). *Experiential learning: Experience as the source of learning and development*. Upper Saddle River, NJ: Prentice Hall.
- Kruchten, P. (2011). Experience teaching software project management in both industrial and academic settings. In *Proceedings of 24th IEEE-CS Conference on Software Engineering Education and Training, Honolulu/Hawaii* (pp. 199–208).
- Lindley, C. A. (2003). Game taxonomies: A high level framework for game analysis and design. Gamasutra Website. Retrieved May 9, 2018, from [http://www.gamasutra.com/view/feature/131205/game\\_taxonomies\\_a\\_high\\_level\\_php](http://www.gamasutra.com/view/feature/131205/game_taxonomies_a_high_level_php).
- Lu, B., & DeClue, T. (2011). Teaching agile methodology in a software engineering capstone course. *Journal of Computing Sciences in Colleges*, 26(5), 293–299.

- Lynch, T. D., Herold, M., Bolinger, J., Deshpande, S., Bihari, T., Ramanathan, J., et al. (2011, October). An agile boot camp: Using a LEGO®-based active game to ground agile development principles. In *Frontiers in education conference (FIE)*, 2011 (pp. F1H-1). IEEE.
- Mahnic, V. (2012). A Capstone Course on agile software development using scrum. *IEEE Transactions on Education*, 55(1), 99–106.
- Malone, T. (1981). Toward a theory of intrinsically motivating instruction. *Cognitive Science*, 4, 333–369.
- Paasivaara, M., Heikkilä, V., Lassenius, C., & Toivola, T. (2014). Teaching students scrum using lego blocks. In *Proceedings of the 36th International Conference on Software Engineering* (pp. 382–391). <http://doi.acm.org/10.1145/2591062.2591169>.
- Percival, F., Ellington, H., & Race, P. (1993). *Handbook of educational technology*. London: Kogan Page.
- Poppendieck, M., & Poppendieck, T. (2003). *Lean software development: An agile toolkit*. The Agile Software Development Series. Addison Wesley.
- Royce, W. W. (1970). *Managing the development of large software systems* (pp. 328–333).
- Schön, D. (1983). *The reflective practitioner: How professionals think in action*. Basic Books.
- Schroeder, A., Klarl, A., Mayer, P., & Kroiß, C. (2012). Teaching agile software development through lab courses. In *Proceedings of the Global Engineering Education Conference (EDUCON)* (pp. 1–10). IEEE.
- Schwaber, K., & Beedle, M. (2002). *Agile software development with Scrum*. Prentice-Hall.
- Scott, E., Rodríguez, G., Soria, A., & Campo, M. (2016). Towards better Scrum learning using learning styles. *Journal of Systems and Software*, 111, 242–253. <https://doi.org/10.1016/j.jss.2015.10.022>.
- Shneiderman, B. (2004). Designing for fun: How can we design user interfaces to be more fun? *Interactions*, 11(5), 48–50.
- Taiichi, O. (1988). *Toyota production system—Beyond large-scale production* (pp. 25–28). Productivity Press. ISBN 0-915299-14-3.
- Wang, X., Conboy, K., & Cawley, O. (2012). Leagile software development: An experience report analysis of the application of lean approaches in agile software development. *The Journal of Systems and Software*, 85(6), 1287–1299.
- Wangenheim, C. G., Savi, R., & Borgatto, A. F. (2013). SCRUMIA—An educational game for teaching SCRUM in computing courses. *Journal of Systems and Software*, 86(10), 26752687. <http://dx.doi.org/10.1016/j.jss.2013.05.030>. ISSN 0164-1212.
- Wangenheim, C. G., & Shull, F. V. (2009). To game or not to game? *IEEE Software*, 26(2), 92–94.
- Williams, L. (2010). Pair programming. *Encyclopedia of software engineering* (Vol. 2).
- Womack, J. P., Jones, D. T., & Roos, D. (1990). *The machine that changed the world*. Free Press. ISBN-13: 978-0-7432-9979-4.
- Wouters, P., et al. (2009). Current practices in serious game research: A review from a learning outcomes perspective. In M. Stansfield & L. Boyle (Eds.), *Game-based learning advancements for multi-sensory human computer interfaces*.

# Red-Green-Go! A Self-Organising Game for Teaching Test-Driven Development



Suzanne M. Embury, Martin Borizanov and Caroline Jay

**Abstract** Teaching test-driven development (TDD) in an already crowded undergraduate curriculum presents a number of challenges. How can we achieve deep understanding of the technique in a limited number of hours, in large classes, with extremely varying programming ability amongst the students and even with wide variations in the prior experience of students with TDD itself? We describe how we have applied a range of agile practices in the design of Red-Green-Go!, a board game for learning TDD that allows students to tailor the learning experience to suit their own level of experience and skill. After unsatisfactory attempts to use traditional teaching methods in our TDD classes, we made use of self-organising teams, big visible charts, frequent feedback and reflection to create a self-paced teaching activity. The game board guides pairs of students through the TDD cycle, as well as introducing students to different pair-coding styles. Feedback is available in various forms, including “community chest” style hint cards, with more extensive explanations and examples on a GitHub wiki for those that need them. Further scaffolding is provided by fully worked examples. As well as describing the agile principles and practices used in the design of the game, we present an experience report from its use over 3 years, with cohorts of up to 100 students per year. We also reflect on the approach we took as a general technique for maximising learning amongst large student groups with widely varying levels of knowledge, experience and skills.

**Keywords** Test-driven development · Agile games · Software refactoring  
Software testing · Big visible charts · Information radiator · Fail fast

---

S. M. Embury (✉) · M. Borizanov · C. Jay  
School of Computer Science, The University of Manchester, Oxford Road, Manchester, UK  
e-mail: [suzanne.m.embury@manchester.ac.uk](mailto:suzanne.m.embury@manchester.ac.uk)

M. Borizanov  
e-mail: [martin.m.borizanov@gmail.com](mailto:martin.m.borizanov@gmail.com)

C. Jay  
e-mail: [caroline.jay@manchester.ac.uk](mailto:caroline.jay@manchester.ac.uk)

© Springer Nature Singapore Pte Ltd. 2019  
D. Parsons and K. MacCallum (eds.), *Agile and Lean Concepts for Teaching and Learning*, [https://doi.org/10.1007/978-981-13-2751-3\\_19](https://doi.org/10.1007/978-981-13-2751-3_19)

# 1 Introduction

As agile approaches become embedded in industrial software engineering practice, the need to include such practices in undergraduate curricula has grown. But teaching agile software engineering in a university context presents a number of pedagogical challenges. It is easy (and efficient) to deliver selected core facts about agile software engineering to large classes using the traditional university lecture format. What is harder to do is to convey the change in mindset needed for agile software engineering, especially when that mindset directly conflicts with the traditional teaching methods themselves. There is an inherent tension in using a standard lecture to convey the idea that, in software engineering, dialogue is a more valuable and effective means of communication than monologue, that “doing” is more useful than “listening” and “observing”, that sometimes the most knowledgeable person in the room is the least experienced or most junior and that control of the communication must shift according to the current goal. How can we embed a genuine understanding of these concepts in the lab and the lecture theatre with the teaching resources available to most university departments?

The agile software engineering community has faced this problem itself, when working out how to prepare software engineers versed in the more traditional forms of software engineering for roles in agile software teams. One solution proposed is the “coaching game” (Parsons, 2014; Paasivaara, Heikkilä, Lassenius, & Toivola, 2014). Coaching games are short, structured activities that place participants in a position analogous to one that could be encountered in an agile team. The game rules guide participants in what actions they can take, while also leaving some aspects open to choice. The game rules then show the consequences of the decisions made by the participants, and allow the team to reflect on the courses of action that led to a successful result and those that led to failure. If the game is well designed, the analogy between the game situation and real software engineering will be clear, and during the debrief participants can reflect on their experiences in the game to deepen their understanding of the core principles being taught.

We have used a number of coaching games proposed by agile participants with success in our undergraduate courses in agile software engineering at the University of Manchester. Games such as the Agile Lego Game,<sup>1</sup> the Specification Game,<sup>2</sup> the 99 Test Balloons Game (McCullough, 2009) and the Ball Point Game (Gloger, n.d.) have all been played in our classes, in some cases with groups of more than 100 students at a time. Feedback from students has been very positive, with at least half of the respondents in end-of-unit surveys reporting that the coaching games helped them to learn the concepts as well as enjoying the active approach to learning.

Where we have been less successful in using coaching games is in teaching the more technical agile practices. In particular, we have not found suitable coaching games to help us teach the test-driven approaches to programming. We have

---

<sup>1</sup>Designed by Sam Newman, Dan North and Mike Hill, at Thoughtworks.

<sup>2</sup>We use the version by Jens Ostergaard, adapted from an original game devised by James Shore.

experienced substantial difficulties in teaching these topics, especially test-driven development (TDD), effectively to our larger classes:

- In a class of 100 undergraduate students, even at final year, there will be a very broad range of programming abilities, knowledge and experience. How do we proceed at a sufficiently fast pace to keep the strong coders engaged, without losing those who are less confident?
- Increasingly, as students are exposed to agile practices and test-driven approaches in industry placements and in their part-time work, classes will also have a very broad range of familiarity with test-driven coding practices. How do we run a single class that deepens the understanding of all students present, regardless of their starting position in terms of TDD knowledge?
- In a crowded software engineering syllabus, we have very limited time (just 2 hours per semester at the time of writing) in which to introduce TDD. How do we convey a meaningful chunk of TDD understanding in this short session?
- It is easy to list the core rules of TDD in a lecture, but applying them in practice is more difficult. How do we steer a class of 100 through the decisions needed to complete several iterations of the TDD cycle, with just one member of academic staff and three teaching assistants (TAs) to steer students back on track when they go astray?

We found a solution to these problems by applying agile techniques to develop a board game for teaching TDD. The game is played by pairs of students, at a variety of different levels. Each pair chooses the level of play to focus on, depending on their particular learning goals, and the level of support and guidance they want to get from the game. These choices can be reviewed and adapted during the lesson, as pairs discover that they need more or less support from the game than they originally envisaged.

In this chapter, we present Red-Green-Go!, the board game we developed to teach TDD, and show how we used a range of agile practices to solve the teaching problems we faced. In summary, we leveraged the following agile practices:

- Self-organising teams: each student pair chooses to play a version of the game that fits their level of ability/experience, and moves forward at a pace that suits their changing level of understanding.
- Information radiators: the game board gives an instant visual indication of where each pair is in the test-code-refactor cycle, as well as the role each pair partner should be playing depending on the style of pair programming being adopted.
- Pair programming: students work in pairs to enable conversations about the problem at hand, and the steps the game leads them through. Thus, students act as the first level of feedback, checking each other's decisions and proposing alternative viewpoints.
- Fail fast: student pairs start to code using TDD right from the first move of the game. Guidance cards kept face down on the board allow students to make their own decisions, while providing on-demand feedback on those decisions, to give an early warning when students are heading off track.

- **Feedback:** to complement the fail fast approach, the game structure offers multiple levels of feedback, which students can choose to access or not, as appropriate to their current learning stage. “Community chest” style cards offer brief guidance in the core TDD and refactoring principles, backed by links to a wiki, containing more in-depth explanations and examples for those who need them. Full worked scenarios give the final level of feedback for teams who want to check the sequence of decisions they made while working through the set problem.

We have used this game to teach three cohorts of students, as part of a final year elective course unit on agile software engineering. Its use has transformed our experience of teaching TDD at undergraduate level.

The remainder of this chapter is organised as follows. We begin by surveying the state of the art in the teaching of TDD and other test-driven methods (Sect. 2). We present the basic form of the game board and how it supports students through the mechanics of the TDD cycle (Sect. 3) before describing the various levels and forms of feedback that we have embedded in the game (Sect. 4). We close by reporting our experiences of using the game to teach undergraduates, including some limitations of the current game (Sect. 5), and draw lessons for the application of this approach to undergraduate level teaching more generally (Sect. 6).

## 2 Teaching Test-Driven Development

### 2.1 Background

Test-driven development is a systematic approach to software programming in which the code for a unit (typically, a class in object-oriented programming) is created gradually, in lockstep with the automated unit tests for the unit. It was proposed by Kent Beck and Robert Martin in the late 1990s, and popularised as part of the eXtreme Programming (XP) methodology (Beck, 2000). The aim is (in Beck’s words) to create “clean code that works” (Beck, 2003), by reversing the order in which the main coding steps are undertaken. In conventional approaches, we first attempt to create an elegant and clean design for the code we plan to write, then we implement it and finally we write tests to check whether it works correctly. In other words, we follow these steps:

1. Design
2. Code
3. Test.

However, as Beck explains, this seemingly logical approach turns out to be more problematic than would at first appear. We start with a clean design, but in the final testing step discover that our code does not actually work. That is, some of the tests fail. The complete implementation exists at this stage, and so fixing the problems revealed by the tests can require a complex sequence of code changes that usually



involves doing some damage to the elegance and clarity of the design we started off with. We end up with code which “works” (i.e., passes the tests) but which is no longer “clean”.

Test-driven development turns this process on its head. Instead of starting by trying to write a clean design, in TDD we concentrate first on writing code that works *and* that is well covered by unit tests. We then make what changes we need to, to clean up the design. We can make these changes with confidence because the unit tests act as a regression test suite, and will tell us (i.e., fail) when we make a code change that breaks our previously working implementation. Thus, coding progresses by undertaking the following sequence of steps:

1. Test
2. Code
3. Design.

As can be seen, this is exactly the opposite of the order in which the tasks are done in the conventional coding approach.

In addition to reversing the order of the steps, TDD requires us to break down our coding tasks into very fine-grained chunks of implementation. Rather than writing tests for the whole of the required functionality in one step, and implementing the complete solution in the next, in TDD we work in units of functionality equivalent to a single unit test at a time. This means that, instead of going through this sequence of steps once per major implementation task, we instead repeat the cycle many times, each time adding a small sliver of additional functionality to the code.

The process works by asking us to first write a unit test that describes an aspect of the requirements that has not yet been implemented. This test should fail when run against the current implementation (which is merely an empty stub at the start of the process). The next step in the cycle asks us to write code that causes the test to pass. More than this, we are asked to make only the simplest smallest code change that will cause the new unit test to pass (while not causing any other tests to fail). In the final step of the process, we examine the quality of the code as a whole, and refactor where we see opportunities to improve and solidify the design, or to generalise the current solution to cover more, similar cases.

Thus, this sequence of steps is actually a cycle that we repeat many times while coding a single unit, in some cases iterating round the entire cycle in just a few minutes. This means that all three tasks (testing, coding and design) must be divided up into a sequence of much smaller tasks. In particular, the design task is converted into a sequence of small-scale design improvements (although periodically it can be necessary to make changes with a broader architectural impact). Such small-scale improvements are known as “refactorings”. Hence, the steps of the TDD cycle are more commonly labelled:

1. Test
2. Code
3. Refactor.

This cycle is also sometimes referred to as the “red, green, green” cycle, after the test results that we expect to see on completion of each step. After the first step, we should have at least one failing test case, which is conventionally shown as a red result in unit testing harnesses. After the coding step, the failing test should pass (and no other tests should have started to fail) so we expect to see the whole test suite passing, conventionally shown as a green result in a unit test harness. Any refactorings we make to improve the design should also not cause any functional regression, and so we again look for a green result as the unit test harness finds that all tests continue to pass.

## ***2.2 The Challenges of Teaching TDD***

The TDD approach sounds simple but in practice requires a significant change of approach compared with conventional coding, and its benefits can remain opaque in the face of the additional challenges it brings for the novice TDDer. To begin with, students need to learn to think about coding problems in terms of a sequence of fine-grained unit tests, rather than in terms of modules and functions. After this, if the unit tests that are identified are too coarse-grained (that is, the student is attempting to code too much functionality in one step), then problems will immediately follow. The coding step will involve adding many lines of code, making it conceptually more challenging and increasing the chance of errors. A too-large coding step can also obscure some of the opportunities for refactoring which need to be applied in order to simplify the subsequent testing and coding steps, as well as making the refactorings harder to apply accurately. Worst of all, if the tests are all coarse-grained, then the test coverage may not be sufficient to catch errors made in the coding step, and these will lie hidden in the code base until some later stage, when they will be much harder to fix.

To add to all this, the sequence in which the unit tests are tackled can make all the difference between a smooth TDD experience and a frustrating one. Unit tests should be chosen in an order that allows the functionality implemented by the unit to grow gradually, in a logical sequence of extensions. This minimises the code changes needed in each coding step and maximises our chances of spotting the refactorings needed to keep the design clean. If tests are tackled in an inappropriate order, then the code unit under development will contain incomplete fragments of different aspects of the implementation, and we will need to keep each strand of the development separate in our mind, in order to refactor effectively. In practice, novice TDD developers in this position simply fail to spot the opportunities for refactoring. The result is that the implementation grows larger, and messier, as the student attempts to graft on the new code needed to make each successive failing test pass. The messy code becomes impossible to understand, so that the student struggles to work out which test cases may still need to be implemented in order to finish. And the clues that would allow the student to refactor their way out of the mess are hopelessly buried.

The importance of the refactoring step in the TDD process is easily missed by students. In fact, refactoring plays two roles in TDD. One of these, the obvious one, is the tidying up of design errors: the renaming of variables once their full role is understood; the combining of cascaded conditional statements into a single more complex conditional; and the wrapping of related groups of primitive variables into value classes, to give a few examples. These are true refactorings in the strictest sense of the word: they change the structure but not the behaviour of the code.

The other role refactoring plays in TDD is to transform code which operates correctly on a select set of input parameters (as defined by the tests) into code which will operate correctly across the whole domain of possible input parameters (or a well-defined sub-domain). This is the kind of refactoring which transforms a code fragment such as the following:

```

if (n == 1) {
    return x;
}
if (n == 2) {
    return x * x;
}
return x * x * x;

```

into the more general form:

```

return x ^ n;

```

Refactorings of this kind do not fit the original, strict definition of the term, since they *do* change the behaviour of the code, as well as its structure. But they preserve the aspects of the behaviour that are verified by the test suite; all the tests that passed before the refactoring still pass after the refactoring. The difference is only in the interpretation of the test suite. Before the refactoring, the test cases describe single point cases. After it, they are representative examples of an equivalence class of tests, standing in for a possibly infinite set of test cases covering a whole domain or sub-domain of input values and their corresponding output values.

The example we have given is over-simplified in order to make the point but is nonetheless representative of the kind of transformation that occurs repeatedly in TDD. The code first grows, somewhat clumsily, as the TDDer follows the strict interpretation of what the tests are asking for, with each new test being followed by the addition of an extra if-statement in the coding steps. Then, when the general pattern is revealed and the refactoring that is required is made plain, the code shrinks, often dramatically, to a more readable and general form.

This kind of refactoring is an essential partner to the TDD coding steps, in which the requirements of the tests are followed exactly and somewhat bloody-mindedly. Without it, the code bloats up in a confusing manner, and never manages to converge on the kind of general solution that is needed. Since most TDD novices start work on very simple examples, the experience can be one of setting out to write code that

would be straightforward to code from scratch using conventional approaches but that turns into a confusing and error-strewn mess when coded using TDD. If the student does not realise that they have missed a key step in the process, they can end up baffled as to what the supposed point of TDD can be.

These problems are compounded by the fact that many undergraduate students are not yet adept at writing automated tests, especially in advance of the code that the tests are aimed at verifying. Nor have they had the chance to develop good refactoring instincts, to be able to spot the code smells that suggest refactorings or to apply them with enough confidence to make the process a painless one. And, in our experience, a significant number of students can find it difficult even to remember what kind of coding step (testing, coding or refactoring) they are supposed to be doing at any one time. All these problems can add up to a frustrating and fruitless teaching and learning experience, for staff and students alike.

### ***2.3 Approaches to Teaching TDD***

Unsurprisingly, there is general agreement that TDD must be tried, hands-on, to be learnt effectively (Vodde & Koskela, 2007; Bravo & Goldman, 2010). How to achieve that in a university environment, with the environment and resource limitations is a more difficult question.

Mugridge (2003) identifies two main challenges in teaching TDD to undergraduate students: the need to change the thinking processes involved in coding and the need to develop skills in testing and design that are often undeveloped in students at this stage. He proposes that fast feedback mechanisms are the solution to these challenges, but notes the difficulty of achieving this in practice.

Some academics have attempted to tackle this problem using different delivery mechanisms, such as coding dojos. In a coding dojo, students volunteer to take it in turns to come and execute each step of the TDD cycle, while the others in the room observe and give suggestions. The coding work is thus done collaboratively, and knowledge is shared throughout the participants. For example, Da Luz, Neto, and Noronha (2013) delivered a TDD class using a mix between a lecture style and a workshop style by adopting the coding dojo approach. A survey of students taking the classes revealed an increase in engagement and interest in TDD after the sessions. Lee, Marepalli, and Yang (2017) undertook a similar study, carrying out two different experiments in which the outcomes for students who attended Coding Dojo style training in TDD were compared with the outcomes for students taught using traditional lectures and labs. They found that code coverage was better for the Coding Dojo participants, and that students in that group viewed the activity as non-competitive and non-threatening.

The coding dojo technique, however, shares some of the disadvantages of a lecture-based style of presentation. In a large class, it is possible for a student never to volunteer, or be selected to implement any of the steps in the process, and it would be easy to disengage and lose focus, and therefore lose the ability to jump in and

contribute code or ideas. Moreover, the whole class must work at the same pace, regardless of the understanding or skill levels of the participants. An extension of the coding dojo idea is the coderetreat, in which participants take a day to repeatedly solve the same coding problem, with different pair partners and exploring different techniques (Parsons, Mathrani, Susnjak, & Leist, 2014). Pairing and TDD are at the heart of this approach, due to their ability to promote conversation and reflection within the coding process.

Other course leaders have looked to tool support to assist in the teaching of TDD. In particular, a number of authors have focused on the difficulty some novice developers experience in writing tests, noting that this is a key barrier for the effective learning of TDD, and have proposed tools to make test writing easier. Miller (2004), for example, proposed the use of a simple text file, containing input/expected output pairs for tests, rather than asking students to learn to code xUnit style tests at the same time as learning to code in general. A very lightweight approach is taken in which the algorithm under implementation is embedded within a programme that executes the code for each input/output pair in the test file.

A more sophisticated approach to the same idea is taken in the ComTest system (Lappalainen, Itkonen, Isomöttönen, & Kollanus, 2010). Students using ComTest can write their tests as Java code embedded within JavaDoc comments in the same file as their solution. An Eclipse plug-in has been developed that allows the tests to be executed, and the results viewed. This approach extends the work of Miller, by allowing test fixture code (setup and teardown) to be included in the specification of the tests, along with more complex input/output specifications. For example, input/output values can be specified as a simple textual table. Wellington, Briggs, and Girard (2007) attempted to make testing even easier for beginning coders, by creating a wizard for test creation, for use within an Eclipse plug-in. The student uses the wizard to select the unit of code under test and supplies the necessary input values and the expected outputs. The wizard then generates the equivalent JUnit code on the student's behalf.

These tools are intended only as a stepping stone to allow beginning programmers to develop the habit of TDD while learning to code, without the added headache of needing to learn a testing harness. It is expected that students will graduate to writing true tests, at some point. The question then becomes how to assess programming exercises carried out using TDD. It is not enough simply to assess the solution as produced at the end of the coding process. We need to know that students have written appropriate tests, and may also wish to assess whether students have followed the TDD rules in order to reach their solution.

Various approaches have been taken to the assessment of TDD-based coding exercises. One solution is to make use of tools common in industry, such as continuous build and test tools in combination with version control systems such as Git. Goodwin and Drange (2015) report the use of T-FLIP, an industry-like coding environment, in which students get feedback on their work through the use of Git and the Bamboo continuous integration tool. The feedback reports on the results of a model test suite prepared by the instructor in advance, and executed against the students' solutions when they are pushed to the Git repository. Feedback is also given regarding student

performance compared with the rest of their cohort. A variety of other tools have been proposed to provide feedback using tests, including Marmoset (Spacco et al., 2006) and Professor CI (Matthies, Treffer, & Uflacker, 2017).

Checking that model tests pass when run against student submissions (and intermediate solutions) can be a useful aid in teaching TDD, especially for assessed work. But test results do not in themselves tell us anything about the quality of the test suite created by students (or its fit to the rules of TDD). Some researchers have proposed tools that can assess the quality of the tests directly. Notably, the Web-CAT system checks the correctness of the tests written by students for correctness by running them against a model solution provided by the investigator, and also assesses completeness of student test suites by measuring the code coverage they give on the model solution (Edwards, 2003). Suleman, Jamieson and Keet (2017) proposed a different approach, in which completeness of student test suites is assessed by running them on mutants created automatically from the model solution.

As can be seen from this survey, while we are beginning to understand the delivery mechanisms and tooling that can help us to teach TDD effectively to undergraduates, a number of open questions remain. Providing students with detailed feedback as they work through hands-on examples seems to be an important part of the solution, but techniques for achieving this *in class* for large undergraduate classes are still needed. In particular, feedback on the combined results of carrying out the steps of the TDD cycle is needed, even if good support is in place for teaching students the skills needed for the individual steps (such as testing and refactoring).

### 3 Information Radiator: The Red-Green-Go! Game Board

The teaching challenges we described in the introduction to this chapter and the preceding section are easy to address when novice TDDers can code in partnership with someone experienced in TDD. The more experienced developer can steer the novice away from potential pitfalls, as well as emphasising key steps that the novice might otherwise miss (such as giving thought to the order in which test cases are tackled, and the need to apply generalising refactorings as well as strict refactorings). Unfortunately, this is not possible in a class of 100 students, taught by 1 lecturer and 2–3 TAs. While it is usual to have between 5 and 10 students with some TDD experience in our cohorts, this is far too few to go around. Moreover, these experienced students will actually have a varied amount of knowledge and understanding, since the term TDD is used loosely by some industry practitioners. In some cases, it can be used to mean merely a type of coding that places emphasis on writing automated tests in some form, either before or after the implementation of the desired functionality. It is therefore risky to rely on students who self-identify as TDD experts to teach other students.

We needed some form of teaching aid that would substitute for a knowledgeable coding partner, as far as possible. Since coaching games had proved effective and popular with our students in other parts of the course, as well as for other aspects of software engineering (Caulfield, Xia, Veal, & Maj, 2011), we set out to develop a teaching game for TDD. The rigid and cyclical structure of the TDD process suggested that a board game might help. Students could place counters on the positions on the board, which would tell them whether they should be writing a test, implementing some code or refactoring. The markings on the board would show the student where to move their counter to, once the current step was complete, and so guide them as to the next kind of coding step to be undertaken.

Here, we are applying the agile practice of the *information radiator*, also known as the *big visible chart*. Rather than requiring students to keep the TDD cycle in their heads, the board and game counters give a clear visual indicator of the current stage of the work, as well as an unambiguous route to the next stage.

At first, we envisaged a simple cyclical board, with three places for counters, corresponding to the three steps of the classic red-green-green TDD cycle, and with arrows indicating the direction of travel. We used colour coding to indicate the test execution result needed to pass to the next stage. However, a prototype made clear that this lacked some essential game-like elements: the starting point was not clear, and nor was the end goal. The game just seemed to loop pointlessly forever.

It also lacked any support for students working in pairs. A secondary learning outcome for us for this session was for students to gain a deeper understanding of the practice of *pair programming* (Williams, Kessler, Cunningham, & Jeffries, 2000). There were several reasons for this. It is a core agile practice that we wished our students to be familiar with, in order to deepen their understanding of the collaboration principles behind the agile approach. We also wished to obtain the educational benefits that come from use of the practice in a classroom situation (McDowell, Werner, Bullock, & Fernald, 2002). Finally, we were concerned that while many of our students were aware of the term pair programming, they often had only a surface understanding of the term.

Like many agile practices, the evocative name of this practice appears to tell you everything you need to know about it: pair programming is when two people work together on a programming task. In reality, there is more to pair programming than just having two people sit at one computer. To be effective, pair programmers must genuinely share the work, swapping who is typing at the keyboard frequently, with the person who is *not* typing working just as intensively as the one who is. Two distinct roles are usually mandated: the *driver* role and the *navigator* role (Wray, 2010). The driver sits at the keyboard and focusses on the detail of the coding task to be completed. The navigator observes the code as typed, and tries to think about the bigger picture and whether the pair is moving in the correct direction overall. Like many agile practices, pair programming converts the business of coding from a solitary activity into a shared one, with conversation (dialogue) as the central means of making progress and checking correctness.

We wanted these same benefits for our students. We wanted the game to encourage students to learn through a conversation about the process of TDD and how it applied to the example they were working on. To maximise this, we needed students to be taking both the driver and navigator roles, and swapping them frequently, rather than having the more confident student hog the keyboard while the less confident student looks on, getting increasingly lost and disengaged.

In our early attempts to introduce pair programming, we found that students really struggled to keep in mind which role they were supposed to be playing. Many were already confused as to which stage of the TDD cycle they were in, and the need to swap pair programming roles as well just added to the confusion. So, we needed our game board to clearly signpost the pairing strategy as well as the TDD cycle.

The board design we came up with is shown in Fig. 1. The board shows two routes, one by road and one by river. Each student in the pair chooses one route and sticks with it throughout the game. The ellipses on the routes indicate places where students should place their counters (we use halma pawns) and pause to complete a task in the TDD cycle. When the student lands on a large ellipse, they must take the driver role, and when they land on a small white ellipse they must take the navigator role. The text on the driver role ellipse indicates which stage of the TDD cycle the pair is in, and so which coding action should be taken. These ellipses are also colour coded; to indicate the result, we expect to see in the test suite when the step is completed.

It can be seen from the game board that the driver and navigator roles switch from one route to the next with each stage in the cycle, following the style of pair programming known as *ping-pong pairing*. This shows students which role they should play at any time and enables them to manage the frequent role shifts alongside the need to remember which stage of the TDD cycle they are in. The instructions for play are given to students on laminated cards (shown in Fig. 2).

The board shows four full TDD cycles, after which point the students will be at the top of the board. This gives a sense of completion and progress for students, who often struggle through their first few cycles. It also gives us a useful checkpoint of team progress when teaching the class, and a point at which a small reward (a chocolate, for example) can be given to keep students motivated. After this point, both routes lead the students back to the bottom of the board for another round of four TDD cycles. Students are often moving more quickly by this point, with more confidence, and the second journey up the board typically takes place much more quickly than the first.

Since we were not guaranteed to have an even number of students in each class, and since we wanted a resource that students could use outside of class should they wish to, we also developed a “solitaire” version of the board, for solo play. This can be seen in Fig. 3. In this version of the board, only the road route is populated with ellipses, and there are no navigator role ellipses. Otherwise, the operation of the board is the same. Other variants of the board can be envisaged for teaching different pairing styles.



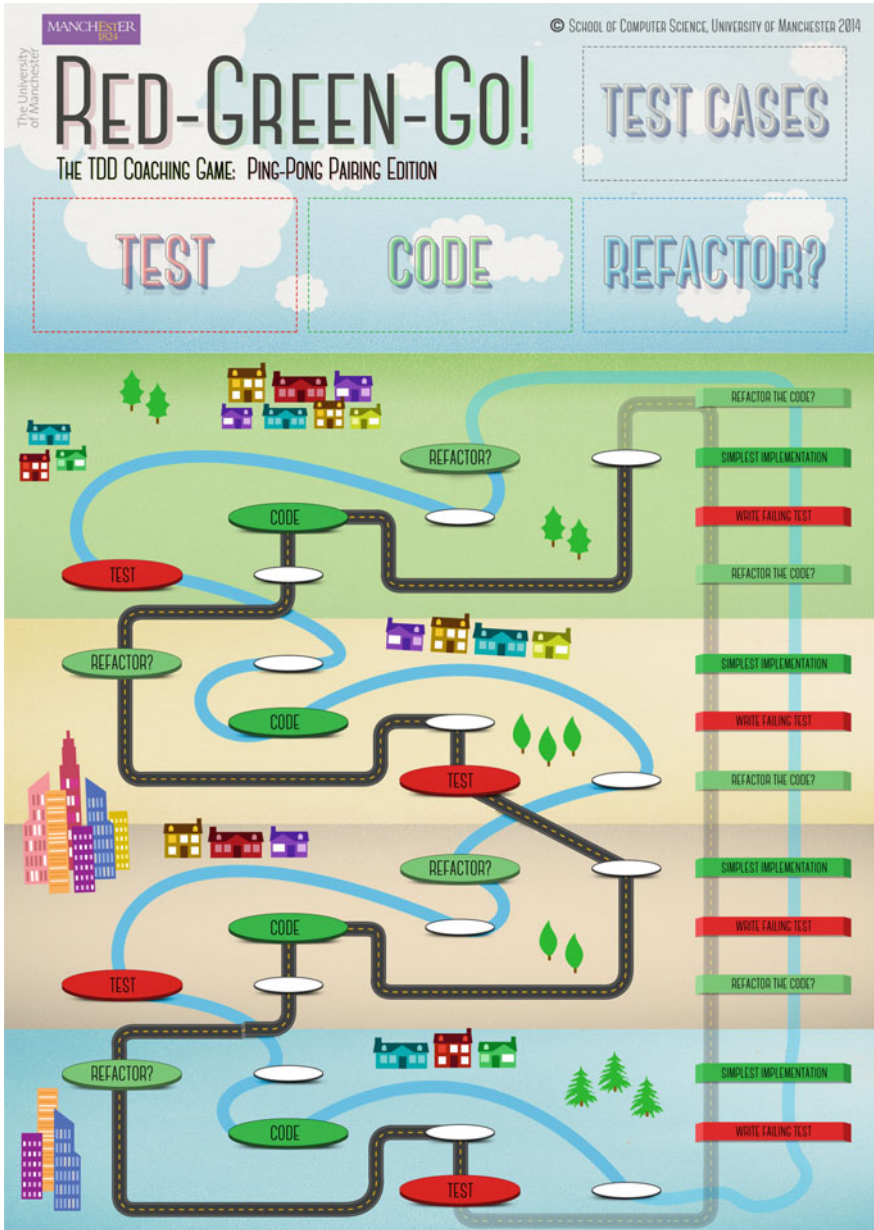


Fig. 1 The Red-Green-Go! Game board for ping-pong pairing

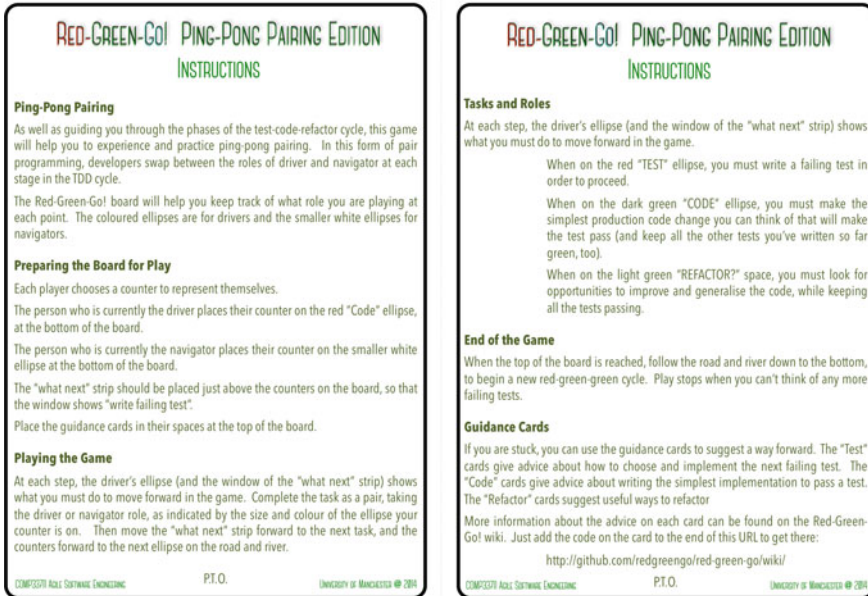


Fig. 2 Instruction card for playing Red-Green-Go!

## 4 Self-Organisation and Trust in Red-Green-Go!

The game board described in the previous section helps students to know what they are supposed to be attempting at each stage, but this will only take them so far. Students also need guidance regarding what is involved in carrying out each step. While we present the basic TDD stages in a short lecture at the beginning of a session, our experience is that students forget most of what they are told when it comes to the business of actually carrying out each stage. On-the-spot guidance is needed to help students get started, and to keep moving forward under their own steam, even when they do not yet fully understand the process they are trying to implement.

To solve this problem, we provided various levels of guidance to students, and allowed each pair to self-organise and set its own goals and starting point. We provide three scenarios that students can choose to work on:

- A quick and easy introductory example covering a small domain and requiring a single method implementation (based on the game Fizz Buzz).
- A simple example covering a bigger domain and requiring a single method implementation (a Roman Numeral converter).
- A trickier example mixing a collection data structure and the need to calculate to a fix point (a text processing example, called Soldiers on Parade).

The scenarios are presented as laminated cards. An example is shown in Fig. 4. Notice that the scenario itself can be played at three different levels, depending on

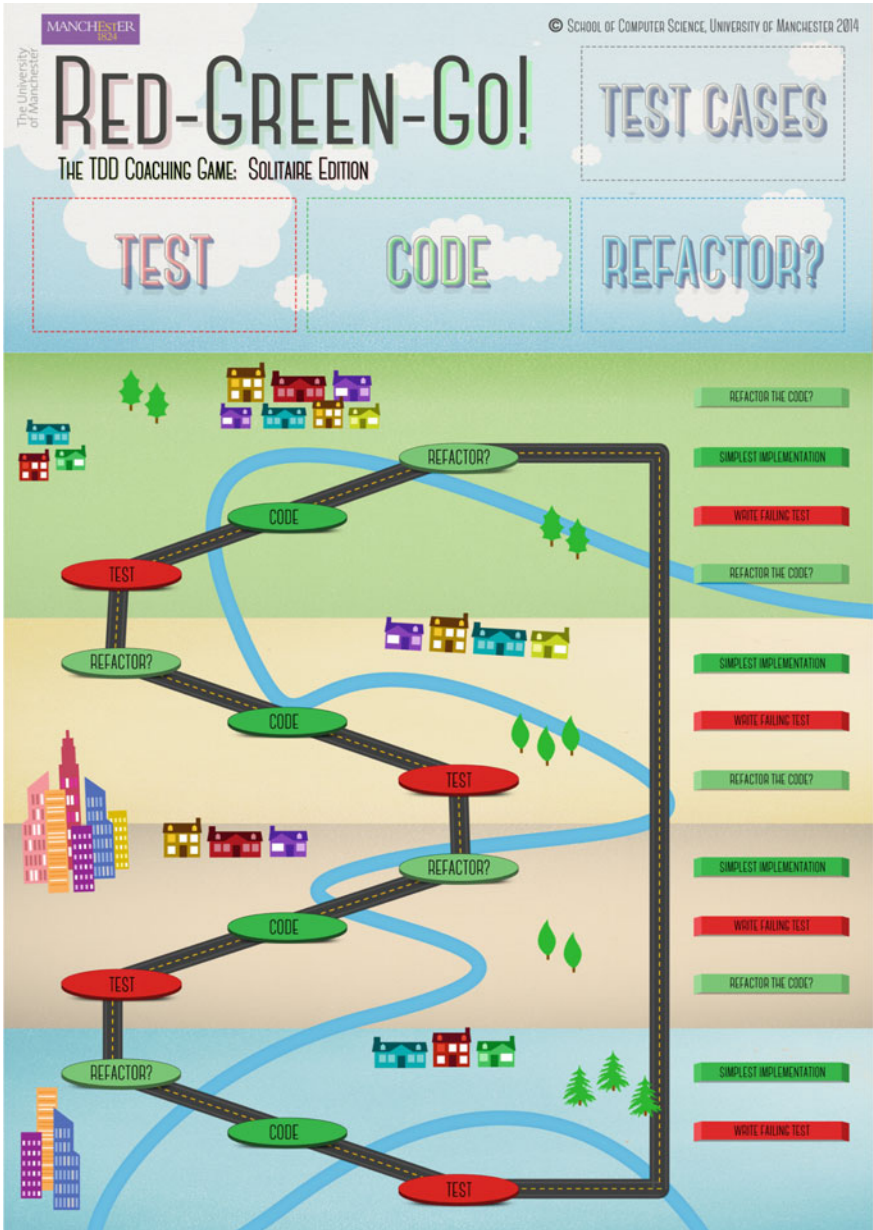


Fig. 3 The Red-Green-Go! Game board for solo play

how much scaffolding students want to get from the materials. Students who are uncertain of how to begin can download a class structure, with stub methods and an

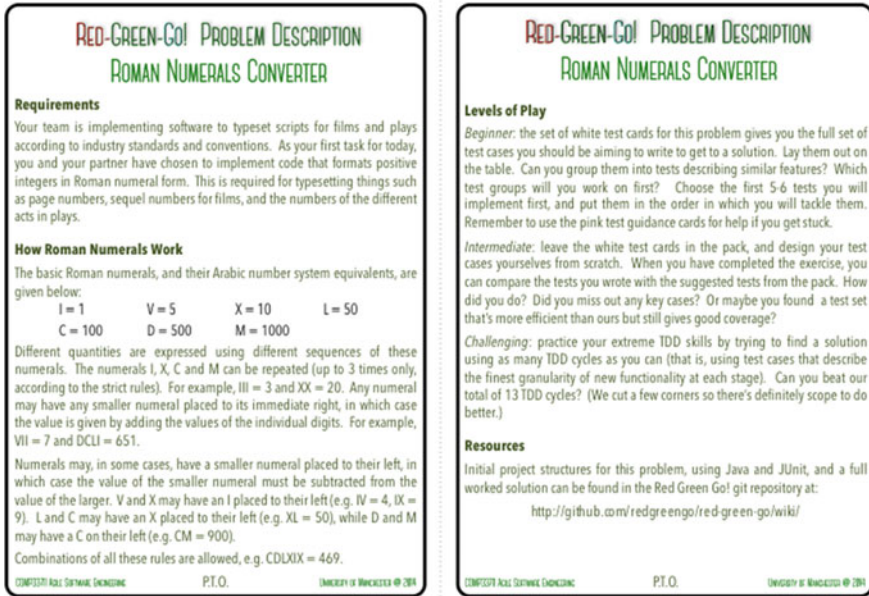


Fig. 4 The roman numerals scenario instruction card

outline test case, from a Git repository to get them started more quickly. Those who are more familiar with test-first coding techniques can design their own production code classes and methods, and write their own tests from scratch.

In addition to the scenarios, we provide “community chest” style guidance cards, giving information and hints about the three steps of the TDD cycle. These cards are placed face down on the three boxes near the top of the game board (labelled *Test*, *Code* and *Refactor*). The idea is that students who are uncertain and can’t move forward can select the top card, read the statement on it and discuss it. If the statement helps, then the students can put the card at the bottom of the pile and start work on their current task. If it does not, they can take another from the top of the pile. Some of the guidance cards for the coding step are shown in Fig. 5 as an example.

As the figure shows, these cards contain only brief, gnomic statements. They can help to jog the memory of a student already familiar with the ideas, or to trigger understanding in a student who is close to that point already, but they will not help the truly baffled. To help in this case, we added a further layer of guidance. Each guidance card comes with an ID code printed on the side (“C1”, “C2”, etc.). This code can be used to locate a fuller description of the concept described by the card, with examples, on the Red-Green-Go! wiki.

Another form of scaffolding we have provided to allow self-organisation of pairs concerns the test cases needed to specify and drive the implementation of the code described in the scenario. At the top right of the board is a box labelled “test cases”. We found that some students experienced difficulties in generating test cases at the

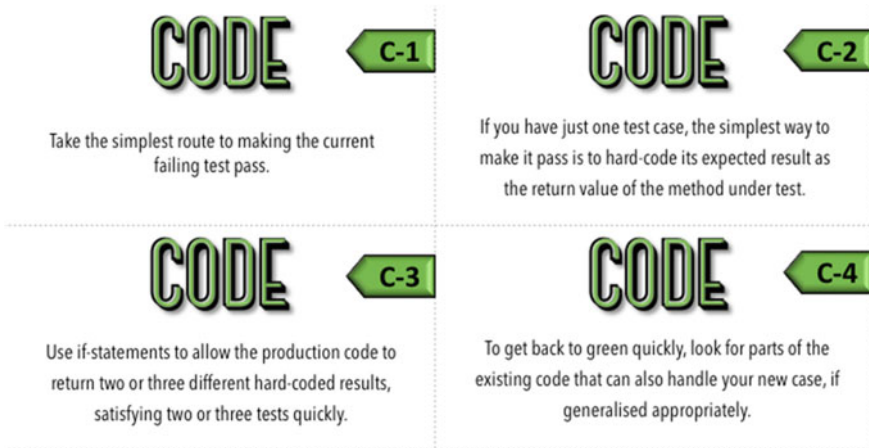


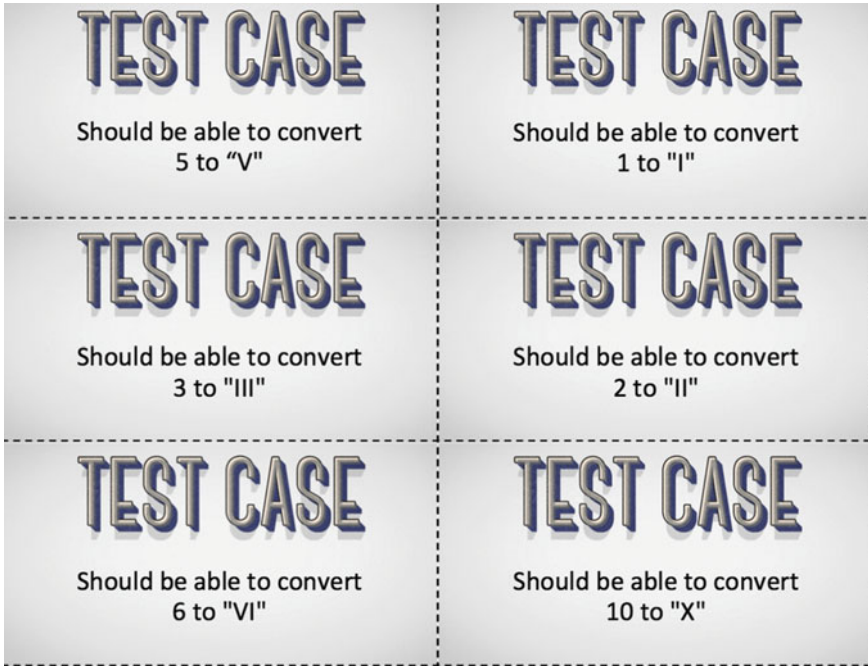
Fig. 5 Examples of the guidance cards for the coding step

correct level of granularity, and that this was a stumbling block for progress with TDD. We also wanted to encourage students to think explicitly about the important question in TDD of the order in which to tackle the test cases. To meet both these needs, we provided with each scenario a set of cards giving the test cases that need to be considered in order to implement the solution with fine-grained TDD steps. The cards are placed face down on the game board. This means that students know they are there, in case of difficulties, but are also first encouraged to try some test design for themselves before they are distracted by the cards. Pairs who are stuck can turn over one or more of the cards and experiment with different orderings before starting to code. A sample of the cards provided for the Roman Numerals scenario is given in Fig. 6.

These cards also act as a form of rapid, on-demand feedback for more confident pairs. Students can first work out for themselves what they think the important early test cases are. Then they can check their answer against the test cases we provided for them. They can compare the granularity of their own tests with the sample solution tests, and make adjustments to their strategy as seems appropriate.

Finally, the Red-Green-Go wiki also contains full worked versions of the scenarios, describing in detail the testing, coding and refactoring steps taken *en route* to a solution, and the reasoning behind each step. They are intended to allow students to check their own solution once it is complete, but also to provide scenario-specific guidance to students who have made some progress, but who become stuck part way through the development.

All these guidance and feedback mechanisms provide a hierarchy of mechanisms by which students can get help to get past blocks in understanding or confidence. The full hierarchy supported by the game, in roughly the order we expect students to consult each item, is as follows:



**Fig. 6** The roman numerals scenario test case cards

1. The game board can give feedback on whether each coding partner is doing the right kind of thing at any one time.
2. The test case cards give help in identifying the right granularity of code increments, and in knowing when the task is complete.
3. The guidance cards give help on carrying out each of the three types of activity (writing a failing test, making a failing test pass, and refactoring to raise the quality level of the code).
4. Advanced level guidance cards cover aspects of TDD that beginners don't need to worry about. Students can choose which level of guidance card they use.
5. The game wiki has additional explanation for all the guidance cards, in case the somewhat terse pronouncements on the cards are not clear or helpful in themselves.
6. Full worked versions of the scenario solutions, showing the steps and reasoning involved, allow students to check their own thinking and provide scenario-specific guidance when the more general resources are not proving useful.
7. Staff and TAs circulate around the room as a last line of defence. Pairs can call for help at any time (though preferably after trying at least some of the other forms of feedback).

The feedback/guidance mechanisms that appear early in the list are quick to access but provide only high-level hints as to the next step needed. Later mechanisms

require more commitment from the students but provide deeper information and more extensive explanation. Staff and TAs are consulted when students feel the cost of accessing the next level of guidance is too high, or when they need help in navigating the choice of guidance mechanism available.

The result is a learning experience in which each pair in the cohort is trusted to set their own goals and to work towards them in the session. The structured nature of the activity means that staff and TAs can zoom in on the particular level/stage of each pair quickly, even when class sizes are large and each pair is working at a very different pace and from a very different starting point. In theory, the activity should scale well to even larger classes than we have tried so far, requiring only a modest increase in the number of TAs as the class size grows.

## 5 Reflections on Red-Green-Go! in Practice

Since the development of the Red-Green-Go! game in the summer of 2014, we have used it with three cohorts of our own students and with one cohort of trainees at an external industry partner. In this section, we report on the experience of using Red-Green-Go! in the classroom and on which parts have been successful and which need further thought.

### 5.1 *The Game Board*

Feedback from students regarding the game board as a way of inculcating the steps in the TDD cycle (and the role swaps involved in pair programming) has fallen broadly into two camps. One group of students has been enthusiastic about the game, found it very helpful and also a fun and engaging way to learn about TDD. The second group failed to see the point of the game, and found it instead an unhelpful distraction. This group tended to abandon the game board early in the session and just concentrated on coding. On our first run of the game, these two groups were roughly equal in size, but in later runs (as we have got better at explaining the point of the game, and as we refined the design of the materials provided to students) this second group has grown smaller in comparison to the first (though still represent a sizeable proportion of the students—roughly a quarter of those who gave feedback in the most recent run of the game).

Students seemed to have few difficulties in navigating the array of choices the game presents them with. Most students started off with very little confidence, and so were happy to go for the easiest scenarios and to make full use of the test case cards. A small number of students each year chose the more advanced scenarios, which have turned out to be important in keeping more experienced students in the room and learning. In previous runs of the course unit, students with TDD experience from their industrial placement year, or from work outside their studies, would steer

clear of any session labelled as an “introduction” to TDD “basics” (labelling which was important for motivating the students without experience). But, even students who have worked on a TDD team for a year in industry can still have a lot to learn, and their presence in the room is important in conveying the value and spread of TDD in practice. Through the advanced scenarios, students with TDD experience can still get a lot out of the session, because it gives them time to reflect on the TDD process they have learnt, to experiment with variations and to compare approaches with students who have worked in other teams and used TDD in different ways.

We also encourage more experienced students to pair with less experienced students, taking a *coaching* role. Coaching is an important part of the agile approach to software engineering, and the game board gives students confidence to take on a coaching role during the session, where in a more free-flow lab-style session, they might be wary of guiding fellow students wrongly. This was a benefit of the game (and particularly the self-organising nature) that we had not anticipated at the start.

The value of the graded scenarios is shown by the way they are used in practice. In each year, we have had a small number of pairs who initially elected to work on the more advanced scenario, only to retreat to the basic scenario after a short time working with it. Students can make this change quietly and without needing to signal any “failure” with the harder scenario. Without this flexibility, inevitably, a portion of students would have been working on a scenario that was too hard or too easy for their current needs, and would feel that the session had wasted their time.

Across all the sessions, we observed that a significant number of pairs (approaching half the class, and including some of those who were enthusiastic about the game) abandoned the use of the game board quite early on. This is not necessarily a problem. The point of the board is to help students form the red-green-green cycle (and the role swapping of pair programming) into a habit that no longer requires conscious thought. If a pair uses the board to learn the basic cycle and starts to feel confident enough to continue without it, then the board has done its job. It was not clear, however, that all pairs who abandon the board are in this happy position. We have noticed a deterioration in the amount of pair role swapping in some teams who abandon the board early, and in the quality of the code produced by these teams. It is impossible to track the detailed progress of all students in these classes, but from the pairs we have been able to observe that it seems that the refactoring step is the one most likely to be neglected when the board is no longer used. Students can successfully operate a test-code-test-code sequence without the board’s help, but often forget to refactor, or else don’t give the time needed to spot the refactoring opportunities present in the code.

## 5.2 *The Guidance/Feedback Mechanisms*

Of the various guidance mechanisms made available to students, the most commonly used, across all cohorts we have observed, are the test case cards. All pairs choosing to work on the easier scenarios will make some use of the test case cards at some



point. Most importantly, the existence of the test case cards forces students to think about the order in which they will work on the tests, in an explicit way, whereas without them students will work on whatever test they can think of next. This aspect of the game has been very successful, and we are planning to increase the number of scenarios with test case cards included. At present, the advanced scenario contains only blank cards, for students to write their own test case ideas on.

The step guidance cards are used by many pairs, especially in the early steps, but are also completely ignored by many pairs. We have observed some teams who do not even place the cards on the board, but leave them to one side. Again, this is not necessarily problematic if pairs are making progress without them. There is the possibility, though, that the cards are not doing their job and that pairs are turning to staff and teaching assistants with queries before looking at the guidance cards. It is possible that the brief cards we have used to date are too aphoristic. They may be good at triggering a remembrance of a concept already understood by a student, but may be much less successful at imparting that concept in the first place.

The wiki pages, containing a more detailed description of the concepts behind the cards, are almost universally ignored by players of the game in our classes. The wiki is hosted in a GitHub repository, and we are able to monitor hits on the pages during the session using the GitHub traffic facility. In some years, there has not been a single access to the wiki pages during the sessions themselves. This was not entirely unexpected. The wiki pages were designed for use by students playing Red-Green-Go! outside the session, using game sets we have made available for students to loan out and play in their own time. Typically, in each cohort, a few students explore the wiki pages in the days following the session, and a handful more do so during the run-up to the exams. But, for the most part, students ignore the additional explanations on the wiki pages. This further raises the question of whether the guidance cards in their current form are sufficient, since it seems that for many students they are the only resource that will be looked at. Maybe a slightly bigger card, that does not sit on the game board itself, and that can contain more information is needed? Maybe students would benefit more from cards that give clues to the specific coding and refactoring steps needed in response to each suggested test case card, rather than the general guidance we currently give? While the solution is not immediately obvious to us, it is clear that this aspect of the game is not providing the benefits we hoped for from it originally and that something different is needed.

The full worked solutions to the scenarios have been used by students, however, and we have received more positive feedback from students about these than about the game itself. These scenarios are intended to be used by students after the session, to reflect on the process as they followed it compared with the TDD steps in the worked solution (and the code produced). For the most part, this is how they have been used, although we have observed a small number of pairs working through the document in the session, when they have got irretrievably stuck while working from the game board and guidance cards alone. The success of the worked solutions, in conjunction with the lessons from the less-successful guidance cards, leads us to conclude that we need to find some way of codifying the knowledge and experience the worked solutions contain, into a form that more easily accessible during gameplay.

### 5.3 *The Lecturer's View*

From an instructional point of view, the game has been a success compared with our earlier attempts at a more conventional teaching approach to this topic. In our earlier attempts, most staff and TA time was taken up by explaining the basic mechanics of TDD. Students were continually uncertain regarding whether they were supposed to be writing a test, or coding, or refactoring. Since few students were following the TDD cycle correctly for even the earliest, easiest cycles, answering these questions for individual pairs proved difficult. Often, they had not followed TDD at all, so working out what stage in the cycle they were at by looking briefly at their code was a futile activity. Telling these students they needed to throw away what they had done and start again from scratch did not make for a rewarding and meaningful learning experience, for anyone concerned.

The introduction of the Red-Green-Go! board game changed this dramatically. An unexpected benefit of the game was the way the game board (in its information radiator role) gives a clear visual indicator to staff of which student pairs are making progress and which are not. In a conventional computer-based exercise, it would not be possible to walk along a row of students sat at computers and understand instantly at which stage each one is at. With the Red-Green-Go! board game, students who are stuck or who are making faster than expected progress are easily visible. Since we also hand out small chocolates each time a pair makes a full circuit of the board, we can tell from the number of sweet wrappers how far a pair is progressing with a scenario. We can also see which scenarios a pair is working on, based on the cards that are placed on the board, and which of the scenario envelopes has been opened.

With the introduction of Red-Green-Go!, all students have been able to follow the TDD cycle for at least a few complete cycles, and the questions about what kind of coding activity needs to be done next have completely disappeared. These have been replaced with questions about the motivation behind TDD, and its strengths and weaknesses: much more productive and useful conversations all round.

The major area of learning that the game does not support well at present is in the refactoring step. Students report that this step is the one where they feel the most uncertainty while playing the game at present, in terms of how to refactor but also in terms of *when* to refactor. Although the game clearly indicates when refactoring should be considered in the TDD cycle, it is often the case that no refactorings are needed in some cycles. It is even possible to refactor too early or too enthusiastically when carrying out TDD, making subsequent coding steps more complicated and future refactoring opportunities (especially for generalising type refactorings) harder to spot. Thus, the refactoring step requires considerably more judgment than the other two steps (both of which are quite mechanical, once the tricky question of the order in which tests will be tackled has been dealt with).

Once again, the mechanics of the Red-Green-Go! board, by keeping students following some approximation of the TDD cycle, allow us to see quickly which pairs are struggling. For pairs who are not carrying out the refactoring step adequately, their code grows in size with each TDD cycle, but decreases dramatically

in quality—without the balancing aspect of the refactoring, students who follow the instructions to “code to the test” in the coding step succeed only in producing large amounts of increasingly horrible code. Success in TDD is vitally dependent on correct refactoring, and this is the aspect of TDD that is least well supported by the Red-Green-Go! game at present. Wider curriculum changes at The University of Manchester are helping with this, in that we are now teaching the writing of automated tests and the skill of refactoring and identifying code smells in the first and second year course units on our undergraduate degree programmes. We will need to observe how these changes affect players of Red-Green-Go! in future cohorts who arrive in our course unit with more of the skills needed for effective TDD than heretofore. But, we should also examine whether changes to Red-Green-Go! can give better support for the refactoring step. One option would be to provide scenario-specific hints (e.g., “If your code for this method grows larger than 12 lines long, you need to go back and refactor”). Keeping the hints scenario specific will allow us to be more prescriptive, and maybe more helpful, than the generic refactoring guidance we give at present.

Even the recognition of this challenge for the design of the game is also evidence of its value, however. In previous years, we would not have been able to gauge the points at which student understanding was being blocked with such precision as Red-Green-Go! allows. The structured nature of the game, while limiting student actions during the session, has freed up both staff and students to consider the deeper aspects of the TDD process. This has been the biggest instructional success of the game: the generation of meaningful conversations about TDD between students, and between students and staff.

#### ***5.4 Limitations/Data Gathering***

The significance and generalizability of the above reflections on the use of Red-Green-Go! in our classes are limited by their anecdotal nature. In assessing the success or otherwise of Red-Green-Go! we faced all the usual challenges of research in computer science education. As Holmboe, McIver and George have stated:

[...] there are obvious difficulties in empirically evaluating [teaching innovations] – aside from the expense of running two concurrent courses and comparing results, such techniques would be ethically dubious, potentially disadvantaging students in one course or the other. Where comparisons can be done across different years, the number of changes between the courses makes it difficult, if not impossible, to evaluate the effect of individual changes. (Holmboe, McIver, & George, 2001, p. 4)

Software engineering academics who wish to gather rigorous evidence regarding the effects of specific approaches to education often fall back on the tried and tested method of issuing questionnaires to their students, after experiencing the innovation in action. But our experience in the course unit supported by Red-Green-Go! is that undergraduate students suffer from survey fatigue, as well as a full timetable. Responses in sufficient numbers to be meaningful can only be obtained by sacrificing

significant amounts of class time. This is a difficult decision for a course leader to make, given that the questionnaire results will not benefit the students who take the time to complete them. Nor does it seem likely that students will be able to correctly assess the depth to which they have learnt a skill in the hours, days or even weeks following the class.

To gather more rigorous evidence for or against the use of Red-Green-Go! in its current form, an alternative approach is to record the actual coding steps taken by students. From this, we could assess how closely students are following the TDD cycle, and how their ability to follow the cycle changes over time. We could ask students to check their code into a Git repository after each small code change, but this adds an additional burden onto students (would we need to include a reminder to commit on the game board?) and seems unlikely to result in records of consistent granularity across cohorts. Alternatively, we could make use of automated monitoring systems, such as that provided by the Marmoset automated assessment system, which captures snapshots of student code on each save (Spacco, 2006). Or, to take this further, we could experiment with the use of the Zorro system, which records low-level coder behaviours within an IDE, such as running a test or invoking a refactoring operation, and which attempts to determine from this whether the users of the IDE are following the TDD cycle or not (Johnson & Kou, 2007).

This could be very informative as regards the learning of TDD, but it will not tell us whether players of Red-Green-Go! are following the correct pairing procedures. For this, we may need to resort to video recording the pairs (with permission) and analysing the pattern of keyboard swaps. However, as always when experimenting on our own students, we will have to consider whether the time taken to gather and analyse the data is worth the resulting gain to future generations of students in the generation of genuinely transferable knowledge.

## 6 Conclusions

Our goal in designing Red-Green-Go! was to enable a class of 60–100 students to be able to experience TDD on a real example, in the space of just 2 hours, with a single staff member and three teaching assistants in support. We wanted a mechanism that would allow students to start doing TDD immediately, even when they do not yet understand what it is or the concepts behind it. This learning-by-doing approach is at the heart of the agile philosophy and continues the theme of asking students to have the courage to “fail fast” in their learning, by trying out techniques even before they have been thoroughly introduced to them.

The game we have designed achieves this by offering several levels of guidance mechanism, from the information radiator of the game board itself, through the various types of guidance card, to the conversations that students have with staff and teaching assistants during and after the session. The guidance mechanisms we have designed are available on-demand to all students, and can also act as a feedback mechanism to allow students to check their decisions at each point, as well as guiding

those decisions in the first place. Further information about the game, and access to some of the game resources used in our teaching, can be found on GitHub at: [github.com/redgreengo/Red-Green-Go](https://github.com/redgreengo/Red-Green-Go).

In our experiences of teaching TDD with the game to three cohorts of undergraduates and one industry team, we have discovered its value as a means of assessing the progress, and points of blockage, of a large class, as well as its value to the students themselves. We have also gained a better understanding of the more challenging aspects of teaching TDD, and of the ways in which our current game design is not providing the support students need. In particular, we need to look at the design of the guidance cards on the TDD steps, and the organisation of the explanatory material in the wiki. We also need to look at ways to help students perform refactoring with more confidence, especially in the key steps towards the middle of each scenario, when important refactoring decisions have to be made. A possible route forward here is in pulling some of the information from the worked solutions (which have proven to be very useful to students) more explicitly into the gameplay.

Our experience with applying the agile practices of information radiators, self-organisation and feedback in the design of this one instructional device points out lessons for the design of teaching materials in general. For us, probably the biggest lesson is the value to be gained from designing teaching activities in a self-paced way, as opposed to a lecturer-guided way. In a self-paced activity, students are provided with the material in a form that allows them to work at their own pace, start at their own starting point, and finish at the point where the learning benefits for them personally start to decline. We have started to move towards this approach with our second year teaching, in which general software engineering concepts are covered, and have been receiving very positive student feedback. These teaching approaches require students to be more engaged, and to take responsibility for their own decisions and for how hard they push themselves in workshops. This is in contrast to the alternative, more passive form of teaching, in which students act as observers while the lecturer guides students (all at the same time) through an activity.

We have also found that taking an information radiator approach to the design of course materials can be helpful in managing a large class of students, all working at their own pace. By considering not only the design of the materials but also the visual manifestation of progress, we can design teaching materials that provide lecturers with minute-by-minute feedback on how student understanding is progressing, at a far finer-grained level than with techniques such as clickers, and in-class quizzes. Of course, not all subjects will be amenable to an information radiator approach, but simple progress charts and scorecards are widely applicable.

In peer review of our TDD workshops for undergraduates by a fellow lecturer, it was commented that our approach requires the lecturer to trust the students to get on with the activity and for the students to have the courage to attempt the exercise even before they had begun to learn about it. Trust and courage to do rather than plan are both key parts of the agile approach to software development that seem likely to have value in any teaching activity to which they are applied.

**Acknowledgments** We would like to thank the members of the Information Management Research group at Manchester who volunteered to play the Red-Green-Go! game in our early trials, and who gave excellent feedback. We would particularly like to thank Iliada Eleftheriou, Fardeen Mackenzie and Leonard Peter Binamungu, for their assistance in running the Red-Green-Go! game in our classes, and to all the University of Manchester undergraduate students who were willing to try the game and to report back on the experience.

## References

- Beck, K. (2000). *Extreme programming explained: Embrace change*. Boston, MA: Addison-Wesley Professional.
- Beck, K. (2003). *Test-driven development: By example*. Boston, MA: Addison-Wesley Professional.
- Bravo, M., & Goldman, A. (2010). Reinforcing the learning of agile practices using coding dojos. In *International Conference on Agile Software Development* (pp. 379–380). Berlin, Heidelberg: Springer.
- Caulfield, C., Xia, J. C., Veal, D., & Maj, S. (2011). A systematic survey of games used for software engineering education. *Modern Applied Science*, 5(6), 28–43.
- Da Luz, R., Neto, A., & Noronha, R. (2013). Teaching TDD, the coding dojo style. In *Proceedings of 13th IEEE International Conference on Advanced Learning Technologies* (pp. 371–375). IEEE.
- Edwards, S. H. (2003). Using test-driven development in the classroom: Providing students with automatic, concrete feedback on performance. In *Proceedings of the International Conference on Education and Information Systems: Technologies and Applications EISTA* (Vol. 3). Citeseer.
- Gloger, B. (n.d.). *The Ball Point Game*. Retrieved from [https://borisgloger.com/wp-content/uploads/2016/08/Ball\\_Point\\_Game.pdf](https://borisgloger.com/wp-content/uploads/2016/08/Ball_Point_Game.pdf).
- Holmboe, C., McIver, L., & George, C. (2001). Research agenda for computer science education. In *13th Workshop of the Psychology of Programming Interest Group* (pp. 207–223).
- Johnson, P. M., & Kou, H. (2007). Automated recognition of test-driven development with Zorro. In *Agile Conference (AGILE), 2007* (pp. 15–25). IEEE.
- Lappalainen, V., Itkonen, J., Isomöttönen, V., & Kollanus, S. (2010). ComTest: A tool to impart TDD and unit testing to introductory level programming. In *Proceedings of the Fifteenth Annual Conference on Innovation and Technology in Computer Science Education* (pp. 63–67). ACM.
- Lee, Y., Marepalli, D. B., & Yang, J. (2017). Teaching test-driven development using Dojo. *Journal of Computing Sciences in Colleges*, 32(4), 106–112.
- Matthies, C., Treffer, A., & Uflacker, M. (2017). Prof. CI: Employing continuous integration services and Github workflows to teach test-driven development. In *2017 IEEE Frontiers in Education Conference* (pp. 1–8). IEEE.
- McCullough, M. (2009). *99 Test Balloons agile game*. Retrieved from <http://tastycupcakes.org/2009/06/99-test-balloons>.
- McDowell, C., Werner, L., Bullock, H., & Fernald, J. (2002). The effects of pair-programming on performance in an introductory programming course. *ACM SIGCSE Bulletin*, 34(1), 38–42.
- Miller, K. W. (2004). Test driven development on the cheap: Text files and explicit scaffolding. *Journal of Computing Sciences in Colleges*, 20(2), 181–189.
- Mugridge, R. (2003). Challenges in teaching test driven development. In *International Conference on Extreme Programming and Agile Processes in Software Engineering* (pp. 410–413). Berlin, Heidelberg: Springer.
- Paasivaara, M., Heikkilä, V., Lassenius, C., & Toivola, T. (2014). Teaching students scrum using LEGO blocks. In *Companion Proceedings of the 36th International Conference on Software Engineering* (pp. 382–391). ACM.

- Parsons, D. (2014). Creating game-like activities in agile software engineering education. In *Proceedings of the Australasian Software Engineering Conference, Education Track, Sydney, Australia*.
- Parsons, D., Mathrani, A., Susnjak, T., & Leist, A. (2014). Coderetreats: Reflective practice and the game of life. *IEEE Software*, 31(4), 58–64.
- Spacco, J., Hovemeyer, D., Pugh, W., Emad, F., Hollingsworth, J. K., & Padua-Perez, N. (2006). Experiences with Marmoset: Designing and using an advanced submission and testing system for programming courses. *ACM SIGCSE Bulletin*, 38(3), 13–17.
- Suleman, H., Jamieson, S., & Keet, M. (2017). Testing test-driven development. In *Annual Conference of the Southern African Computer Lecturers' Association* (pp. 241–248). Cham: Springer.
- Vodde, B., & Koskela, L. (2007). Learning test-driven development by counting lines. *IEEE Software*, 24(3).
- Wellington, C. A., Briggs, T. H., & Girard, C. D. (2007, August). Experiences using automated tests and test driven development in computer science I. In *Agile Conference (AGILE), 2007* (pp. 106–112). IEEE.
- Williams, L., Kessler, R. R., Cunningham, W., & Jeffries, R. (2000). Strengthening the case for pair programming. *IEEE Software*, 17(4), 19–25.
- Wray, S. (2010). How pair programming really works. *IEEE Software*, 27(1), 50–55.