Teaching Agile Methods to Software Engineering Professionals: 10 Years, 1000 Release Plans

Angela Martin[1], Craig Anslow[2](✉), and David Johnson[3]

[1] Xero, Wellington, New Zealand
angela.m.martin@gmail.com
[2] School of Engineering and Computer Science,
Victoria University of Wellington, Wellington, New Zealand
craig@ecs.vuw.ac.nz
[3] Oxford E-Research Centre, University of Oxford, Oxford, UK
david.johnson@oerc.ox.ac.uk

Abstract. Agile methods are an essential resource for software engineers. The Agile movement evolved out of industry and is the common approach to software development today. Teaching Agile methods challenges students' working attitudes, where putting Agile into practice is not possible through simply applying methods prescriptively, but by having an Agile mindset. In this paper we present and discuss our experiences over the last decade of teaching a novel intensive Agile methods week long course as part of a professional Masters of Software Engineering degree programme at the University of Oxford. We describe the typical shape of the course, discuss how students experience Agile values and management practices to foster an Agile mindset, and provide student feedback indicating a consistently positive response to our approach at teaching Agile methods to software engineering professionals. Our reported experiences and material can help other educators who want to run similar courses and adapt where required.

Keywords: Agile software development · Experience report · Group work · Graduate programs · Software engineering professionals

1 Introduction

Since the introduction of the Agile Manifesto, Agile methods in software engineering have gained popularity year on year, and today Agile is not just commonplace, but often expected as a standard industry practice in software development teams. Agile methods were evolved by and are applied by industry [6]. This growth in applying Agile principles in the software industry went hand-in-hand with a growth in Agile training being offered to software engineering professionals in the work place, as well as more recently in undergraduate and some graduate computer science and software engineering degrees.

The University of Oxford Software Engineering Programme (SEP) was established in 1993 and exists to create strong connections between theory and practice in software engineering and to make the expertise of the university available to those who wish to study part-time while continuing in full-time employment. Most students on SEP are practicing software engineering professionals who often already have university degrees or extensive industry experience. Week-long intensive courses in a variety of subjects are offered, with up to 16 students per class. Each student must take 10 courses in any order over a four year period and are used as credit towards a Masters' degree (MSc) in Software Engineering awarded by the University of Oxford. In 2007 the Agile Methods (AGM) course[1] was introduced in response to the growing needs for software engineering professionals to understand and introduce Agile in their work places.

Teaching Agile methodologies often focuses on learning a particular method [6], such as Scrum [14] or XP [2]. It was recognized that the intensive nature of the week long part-time courses at Oxford made it difficult for an in-depth dive into the variety of Agile methods and practices to fit into the short time span of an individual course. To this end, the Agile Methods course is devised to bring students into an Agile mindset – through a combination of (1) coupling lectures with simulated exercises of Agile management practices, (2) critical analysis and debate around case studies on Agile adoption, and (3) hands-on approach of Agile practices within the classroom.

In this paper, we present an approach to teaching Agile to software engineering professionals and discuss our experiences over the last 10 years of delivering the course. We give a course outline describing the pre-course assignment, case studies, lecture content, group exercises, post-course assignment, and finally discuss lessons learned from teaching the course over a long period of time. Other educators who wish to run similar courses can learn from our experiences and material reported in this paper and adapt where required.

2 Course Outline

The Agile Methods course aims to give an overview of Agile to software engineering professionals and help them understand and adopt an Agile mindset. The learning objectives of the course are as follows: (1) compare and contrast the different agile methods, (2) determine the suitability of agile methods for a particular project and organization, (3) evaluate how well a project is following agile principles, and assist the project to become more agile (where appropriate), (4) understand the relationship between the customer and the development team in agile projects and the responsibilities of both communities, and (5) how to foster organizational change to build better software.

The course is scheduled for a week and spans five consecutive days (Monday to Friday), where each day is timetabled from 0900 to 1730, except for Friday where the class concludes at lunch time (see Table 1 for an example Agile Methods course schedule). The week long class is split into discrete time boxes, with three sessions in the mornings, and three in the afternoons, concluding each day with a learning stand up. Each time box consists of a lecture, an exercise, or a case study discussion, with breaks in between each session.

[1] https://www.cs.ox.ac.uk/softeng/subjects/AGM.html.

Table 1. An example University of Oxford Agile Methods (AGM) course schedule. Encoding: Lectures – yellow, Group Exercises – blue, Case Study – green. Three sessions in the morning and three in the afternoon followed by a learning stand-up. Small coffee and tea breaks happen between each session.

Time	**Monday**	**Tuesday**	**Wednesday**	**Thursday**	**Friday**
0900-1000	Introduction	Case Study	Case Study	Case Study	Case Study
1015-1115	Agile Manifesto	XP	Empirical Research	Personas	Retrospectives
1130-1230	Communication	Release Planning & User Stories	Lean & Kanban	User Stories	Retrospective Q&A, Survey
1230-1330	*Lunch*	*Lunch*	*Lunch*	*Lunch*	*Lunch*
1330-1430	Case Study	Case Study	Case Study	Estimation	
1445-1545	Scrum	Iteration Planning & Estimation	Kanban Game	Release Planning	
1600-1700	Marshmallow Challenge	Coffee Machine Game	Kanban Game Cont'd	Iteration Planning	
1700-1730	Learning stand-up	Learning stand-up	Learning stand-up	Learning stand-up	

The only prerequisite is that a student must be already enrolled in the SEP. To cover enough Agile background material and different methods we use Agile Software Development Ecosystems [8] as the text book. A pre-study assignment is given to each student to help them prepare in advance of the teaching week and a post-course assignment as the student's assessment.

2.1 Pre-study Assignment: Case Study

It is important for students to begin their study about Agile in advance of the teaching week, and we cater for this by sending them an assignment four weeks in advance of the course. One of the main themes that the course explores is that of Agile adoption; and not just the idealized version of Agile adoption, but the in-the-trenches realities of Agile adoption. We incorporate a case-based learning approach which is common in MBA programs [7].

Students are assigned a case study on Agile adoption and prepare a short presentation to be delivered during the class, followed by a mediated class discussion. Table 1 highlights the case study presentations schedule in green and Sect. 6 lists the case studies for the 2016 course editions. The case study papers are Agile adoption experience reports from past XP and Agile conferences. The case studies involves students actively discussing different industry-based case studies that focus on different organizations that go through an Agile adoption process. No single case study describes an easy Agile adoption story; each highlights a different discussion point around adoption or organizational change.

Each student presents their case study once throughout the week for up to 30 mins. The student summarizes the paper and leads a discussion. The students are asked to first present on who the organization is, who the author is and what their role is within the organization, and what they are trying to achieve or improve in the organization. The class then discusses what should the organization do based on this information. The presenter then describes what the authors actually did and what the outcome of the case study was. Finally, the class discusses if what the authors did made sense, if something different should be recommended, and compares this study with other case studies that have already been presented.

This case-based learning approach enables students to gain an appreciation of how difficult Agile adoption is at an organizational level. By discussing a range of case studies we aim to equip students with knowledge of a broad range of situations that may arise and be able to think critically about where Agile methods can and should be applied in practice.

2.2 Lectures

During the week lectures are delivered that fit into one hour time boxes and consist of presentations and class exercises. Throughout the week we disperse the lectures among case study sessions and group exercises, to keep class activity varied and to ensure the theory is backed up with practical exercises. Table 1 highlights the lectures in yellow.

Agile Manifesto. After an introductory lecture, we present the Agile manifesto and the 12 underlying principles. We focus on the main idea of the manifesto that is: *We are uncovering better ways of developing software by doing it and helping others do it.* We emphasize the importance of the main items of the manifesto, discuss the principles of the manifesto and give some examples to illustrate these principles and values. Finally, we finish the lecture with some of the common misconceptions about Agile methodologies and from our own experience such as *If you're going to adopt Agile development, you should do it 100%* and *Switching to Agile development offers excellent immediate benefits.*

Agile Methods. We present lectures on time-boxed methods in Agile, where we give an overview of both Scrum and XP. For each method we give an overview of the main features, the different practices and roles that team members have, and explain the core values and contributions. In both methodologies we focus on explaining delivering business value with regular steps, monitoring features delivered, and adjusting plans according to results. Then we discuss balancing allowing the business to change their mind while the development team continues to get work done on a stable scope. We present the different team roles, different practices such as sprints/iterations, maintaining a product backlog, planning, daily meetings, and iteration reviews. We emphasize the values of Agile teams: commitment, focus, openness, respect, and courage. Scrum and XP have similar and overlapping structures, roles, and values. There are however some subtle differences that we highlight in the Scrum and XP lectures, for example where XP has a greater emphasis on engineering practices such as pair programming and Test-Driven Development (TDD). We feel it is important to cover both of these time boxed methodologies, as Agile training frequently champions one method over the other. Our approach is to give students an understanding of what Agile methodologies are available to them, with a view to helping them to think in an Agile mindset and not focus on just the methods. We additionally give overviews of other methods including: Kanban, Lean, Crystal, DSDM, and CRISP-DM.

Release and Iteration Planning. Understanding user stories is a an important aspect to software release planning in Agile. Not only are they used to elicit requirements from customers and communicate ideas among a team, they are used as units of customer value. In Agile, delivering customer value is a priority, and by creating user stories teams can plan releases, as well as iterations, around maximizing value for their customers. We present lectures on how to generate personas (fictional end-users as a focus for delivering value to somebody) and use them in-turn to generate candidate user stories. We then show how to estimate the amount of work a user story might require to be implemented. One of the key things we try and get across is that user stories are not all equal, and that an estimation of the amount of work required to implement one varies from team to team. Estimation is difficult, and requires team discussion and agreement, and we illustrate this idea with students playing Planning Poker[2], among other methods, to estimate animal points (see Fig. 1(a)). We then show how user stories with estimates can be used to plan releases (e.g. a release after 4 week sprints) by selecting a series of user stories that delivers minimum demonstrable value for customers (in order to receive feedback in as short a time as possible). At the same time iteration planning (e.g. weekly) is discussed to show how an Agile team should aim to have working software as early as possible and often. Coupled with the team release planning exercise, our aim is to put students through the motions of becoming customer-focused and in the mindset of team collaboration to achieve goals in an iterative manner. We also discuss how to effectively track progress during release and iteration planning using various techniques such as information radiators (e.g. burn down/up charts).

Guest Lectures. We typically invite expert guest lecturers (industry practitioners or other academics) to deliver specialist content. In particular we have had lectures on Example Driven Development (xDD) (e.g. TDD, ATDD, BDD), Lean & Kanban, change management, and empirical research on how Agile methods are used in practice.

Retrospectives. The final lecture is on retrospectives, where we typically have a guest lecture present and then perform a retrospective exercise with the class. This lecture focuses on the ideas from Derby and Larsen [5] and Kerth [9]. Once the retrospective has completed, the post-course assignment (Sect. 2.4) is explained and handed out and then finally students conduct a course survey which is used for course evaluation purposes.

[2] http://www.planningpoker.com/.

2.3 Group Exercises

One of the key approaches we take with teaching our Agile Methods course is to encourage peer learning and learning-by-doing. To reinforce the lectures students participate in a number of exercises, working as pairs and groups. Table 1 highlights the group exercises in blue.

Communication. We explain to the students how important it is to communicate effectively with customers on Agile projects by illustrating the customer design cartoon[3]. To reinforce this message the students complete a communication exercise – Offing the Off-Site Customer [4]. This exercise involves pairs of students where one acts as a "customer" and the other as a "programmer". The aim of the exercise is for the programmer to elicit requirements from the customer in order to draw a diagram of the product vision on paper (see Fig. 1(b)). The customers are given a drawing that they need to communicate to the programmers to recreate, without visually communicating the drawing. The exercise is played out over two rounds. In the first round the customers must only communicate with the programmer via handwritten text messages on index cards. In the second round, the customers and programmers are allowed to use verbal communication. After each round, the students reflect on the experiences of trying to communicate the drawings between them. The point of this exercise is to suggest that people in close proximity to each other with minimal physical barriers have a better chance of communicating effectively. We encourage the students to think about their own work place and to find a way to set up environments to encourage regular and meaningful collaboration.

Prototyping. To get a feel of prototyping with time-boxed methods, students complete the Marshmallow Challenge[5] and are asked to prototype a fully functioning coffee machine out of cardboard based on ideas from Cockburn [4] (see Fig. 1(c)). The goal of the coffee machine exercise is to try Scrum for an iteration and then walk a mile in a product owner's shoes as part of a second iteration. The students are divided into teams (up to four). During the first iteration the teams have 7 min to write stories about how the machine will work and prepare materials (e.g. boxes, tape, scissors, cups, water). For each iteration they have 5 min to plan what stories they will implement, followed by 10 min to design and implement the stories, and finally a group presentation to the class. The purpose of this exercise is to get the students to work in a simulated environment as a team in a time-boxed manner, and to understand that prototyping and early releases only need to demonstrate a concept to a customer to deliver value.

Agile Debate. To help students gain a better understanding of the differences between the Scrum and XP methodologies we ask them to perform a debate. The class is separated into two sides: Scrum and XP. We give them two well-known,

[3] http://projectcartoon.com.
[4] http://www.jamesshore.com/Presentations/OffingTheOffsiteCustomer.html.
[5] http://www.tomwujec.com/design-projects/marshmallow-challenge/.

and highly contrasting, quotes, from Martin Fowler[6] and Ken Schwaber[7] as the basis for their arguments. The students use the session to come up with their own arguments and perform the debate with the lecturer as the adjudicator. The aim of the debate is for the students to understand that there is no silver-bullet when it comes to applying and adopting Agile methods.

Kanban Game. To help students gain an appreciation of the Kanban method we get them to play the getKanban[8] board game (see Fig. 1(d)). getKanban is a physical board game designed to teach the concepts and mechanics of Kanban for software development in a classroom setting. Each team can have up to six people. Each team has a playing board representing a Kanban task board, and a collection of story cards representing work to be done. Teams compete to maximise profit by optimizing the flow of work. We simulate the game for up to 21 days. During the game the teams construct charts based on data from the game including a Cumulative Flow Diagram, a Run Chart, and a Lead Time Distribution Chart. To help make the game more realistic there are a number of simulated events that occur throughout the game that challenge the teams (e.g. a developer needs to attend a training course) and require them to make various system design, prioritization, and resource allocation decisions. We allow a couple of hours to play the game and have a debrief session at the end to help students understand the intricacies of the method.

Team Release Planning. Building on the accompanying lecture sessions, students carry out a team release planning exercise which covers most of Thursday. This puts into practice everything they have been taught about Agile. In this exercise, we split the students into groups of no more than four per team. In this exercise, we do not mandate any team structure – we allow the students to self-organize, much like a real Agile team would be expected to do. The lecturer sets a particular domain area (e.g. solve London's transport issues) in which each team can then pick their own idea for a small product or service. They create a release plan (including personas and user stories) over four weeks and four time-boxes on a card wall, with the aim of being able to release a first version of their product to a customer after the four weeks. At the end of the exercise, each team presents their release plan to the rest of the class (see Fig. 1(e)).

Retrospective. A retrospective on the course is performed on the last day where an external guest lecturer usually facilitates. Students record their thoughts about the course on post-it notes into three categories: positive, could be better, and aspects that were a surprise. Students place the ideas into different days of the course on a card wall based on the timetable (see Fig. 1(f)). The facilitator walks through the card wall and identifies and discusses key themes. The aim of this exercise is for students to reflect upon what they have learned.

Learning Stand-up Meeting. At the end of each day the students perform a learning stand-up meeting similar to a daily stand-up meeting (see Fig. 1(g)).

[6] http://martinfowler.com/bliki/FlaccidScrum.html.
[7] http://kenschwaber.wordpress.com/2010/06/10/waterfall-leankanban-and-scrum-2/.
[8] https://getkanban.com/.

(a) Estimation: planning poker with animal cards.

(b) Communication: customers & programmers diagrams.

(c) Prototyping: cardboard coffee machine.

(d) getKanban board game.

(e) Team Release Planning Exercise.

(f) Retrospective on the class.

(g) Learning Standup that happens at the end of each day.

Fig. 1. Agile methods course – some sample class exercises that simulate some key points about different Agile practices including: estimation, communication, prototyping with cardboard, Kanban, release planning, class retrospectives, and daily learning stand up meetings.

The stand-up meeting is for each student to address the questions like they would in a daily stand-up, hence fostering Agile team values of openness, respect, and courage. The questions focus on what have they learned during the day and what they would like to learn. Students write answers on post-it notes, present them to the group, and then put them on a learning card wall.

2.4 Course Assignment: Essay and Release Plan

The course is assessed with an assignment where students are given six weeks to develop a mock four-week release plan and complete an essay. The *release plan* is based on a fictitious product idea (e.g. develop an application for a hospital to help support children who suffer from a medical condition like autism). The release plan is documented as a report, outlining the different personas, user stories (based on one persona), and the release plan itself, similar to the team planning exercise performed in class. They need to include the rationale for deciding the team's capacity for each sprint, and why they think that this release plan makes the most sense for the customer. Developing the release plan in the assignment aims to assess what the students learned in class, and we ask them to reflect on this aspect comparing with their experience on the team release planning exercise. The *essay* involves comparing and contrasting different Agile adoption paths from two of the case studies (Sects. 2.1 and 6). One question that students are asked to address is, "is there a one-size fits all Agile adoption strategy?" The assignments are assessed following a marking guide[9] where all submissions are awarded a numerical grade between 0 and 100, interpreted as follows: 0 and 49 denotes a fail, 50 and 69 denotes a pass, and 70 and 100 denotes excellence. Most students are awarded a pass, some with excellence, and few with fail; and grades are released approximately six weeks from submission. Students can defer submitting the assignment and wait for a later edition, but this is rare.

3 Discussion

Teaching the AGM course over the last decade has given us in-depth experiences from which to draw upon, that we would like to share. Throughout the duration of this course, we have gathered student feedback to help inform and evolve the course along the way, as well as having gathered formal feedback from students, some of which we now discuss.

Lectures. One of the biggest challenges in designing this course is catering for the intensive nature of the course delivery. While there are just over 30 h of face-to-face time allocated to the course, we were conscious not to overwhelm students with only lectures. To this end, about a third of the class time was allocated for lectures, broken up with exercises and case study discussion sessions. While lectures are useful for delivering information to students, our key aim was to enable the Agile mindset. In this case, we put further emphasis on learning-by-doing with hands-on exercises and class discussion. For each lecture, we ensured that a relevant exercise or case study followed. Keeping the lectures to a planned limit of only one hour per session we felt also deferred any feelings of fatigue or boredom, which is vitally important in a short but intensive learning environment. The students found the theory was important to learn and appreciated the lecture content on empirical studies of Agile project teams which showed evidence about the use of different practices within industry.

[9] http://www.cs.ox.ac.uk/softeng/handbook/examinations.html.

Group Exercises. The exercises were carefully chosen to help the students put into practice, or to help them quickly understand, the lecture material. As discussed above, the lectures were normally paired with relevant exercises or case studies. For example, to take the learning from the release planning lecture and put it into practice, we would follow the lecture with an exercise on story estimation. Each of the exercises aimed to teach important aspects about the different Agile methods. The communication exercise highlights the importance of fast and frequent customer feedback. The prototyping exercise looks at how Agile teams are formed and how to respond to change. The estimation exercise put the use of abstract story points into practice on non-software artifacts to help understand that estimation is a team effort and not a formula that is uniformly applied. The Kanban board game exercise aims to demonstrate the Kanban method and allows students to put the method in action to understand workflow.

Team Release Planning Exercise. The team release planning exercise involves putting into practice the learning from all the previous lectures and exercises. We encouraged the students to self-organize and gave them reminders and guidance on the course material. We mainly left them to make their own decisions on how to plan their product releases. This exercise always proves to be the most challenging for the students, as it was designed to simulate the real planning of a hypothetical product or service, where the exercise often took many hours or a whole day. The task also requires a level of creativity that many students were uncomfortable with, but it was essential to move them away from their comfort zone in order to get into an Agile mindset where all members of a team should be able to feel they can contribute.

Assignments. The assignment tasks mirrored much of what was taught during the class, where students are asked to write a short essay comparing and contrasting two case studies, and then to create a release plan similar to their team release planning exercise. The essay question was generally straight forward as the case study presentations and debates prepared students well for this part of the assignment. The release planning exercise, however, proved challenging for many. The main difficulty in this task was that in class it was done as a group exercise, while in the assignment the students were asked to do a similar plan but on their own. Some students took the initiative to simulate the group environment by asking work colleagues to carry out the collaborative parts of the exercise, such as estimation. Others on occasions, however, fell back into old habits or forgot the learning in class and strayed on a tangent to what was expected. The creativity aspect of the release planning exercise, on occasions, proved problematic, where students either could not come up with an idea for a product that was suitable to generate a good number of personas and user stories, or that sometimes a student would get carried away and produce an assignment submission that was all about their great idea, but little in substance for demonstrating what they learned in class. What we learned is that setting such an assignment, care should be taken to ensure the students remember they need to focus and demonstrate their learning in their submissions.

Practical Approach to Teaching Agile. The design of the class delivery gives students a practical approach to an Agile environment. The time-boxed class sessions planned throughout the week reflects how sprints or iterations are planned in time-boxed Agile methods such as Scrum and XP. We used pair-stairs[10] to encourage students to pair with their colleagues during the week, much like when applying the XP practice of pair programming. The daily learning stand-up reflects the practice of daily stand-up meetings in a Scrum or XP team. The use of visual cues around the classroom to learning material, but also to the collective class experiences such as in the prototyping and the release planning exercises, provides the tactile experience that Agile working environments give. Finally, getting the students to perform the retrospective exercise gives them the experience of participating in a realistic retrospective.

Evolving Agile Teaching Content. One of the things we have observed that is changing in the students over the last few years is that more students attending the course identify themselves as already having Agile training, or as being Agile practitioners in their organizations. This reflects what we see today in the software industry – that Agile is no longer a niche, and is an expected workplace practice in software engineering teams. To this end we have taken the opportunity, through the retrospectives, to be Agile in our planning of the course itself, and to take on board "customer feedback" from our students and make continual changes based on this feedback. For example we have introduced new games and techniques into the course such as the getKanban board game and used more recent and up to date Agile adoption case studies over time.

Student Feedback. Over each iteration of the Agile Methods course, as discussed earlier, feedback is gained by putting students through a retrospective exercise at the end of the teaching week to help inform any changes to the next instance of the class. Alongside this, students have been returning student feedback questionnaires since 2010, where overwhelmingly the feedback has been positive. The students were asked to rate between 1 (strongly disagree) and 5 (strongly agree) their level of agreement on 12 statements, for example:

– The lectures added significant value to the course material
– The lectures included valuable contributions from other students in class
– The exercises helped me to understand the topics covered in the lectures

The aggregate and average score over the time period for which we have data is 4.32 out of 5 based on 132 completed questionnaires[11] (see Fig. 2). The last three editions of the course feedback scored the most highly and were above the SEP all courses average of 4.55, at 4.6, 4.73, and 4.66 respectively. This feedback shows some perceived evidence that the course structure and content has matured where students are generally satisfied with what is being delivered.

While the feedback was generally positive, there were however some negative comments on the course content in the feedback questionnaires. Many of these

[10] http://pairstair.com.
[11] Statements and raw data located in https://tinyurl.com/AGM-Student-Feedback.

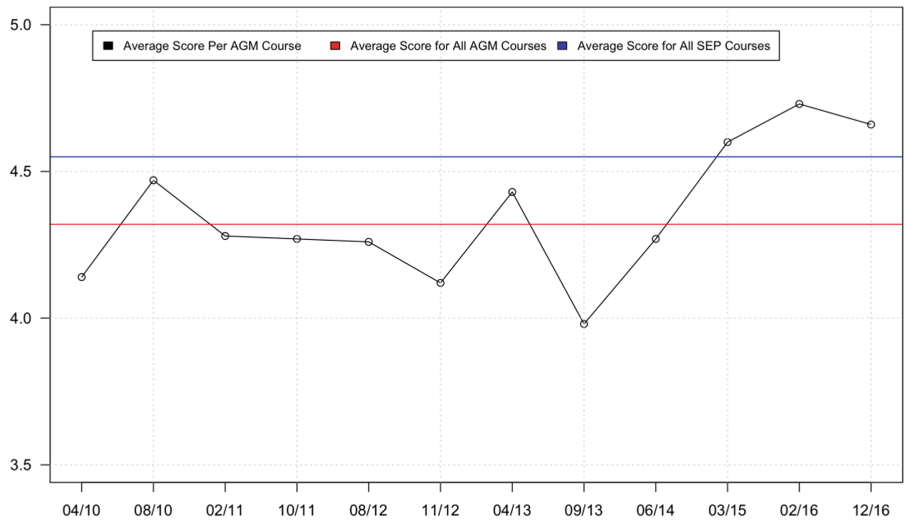

Fig. 2. AGM course evaluation as perceived feedback by students since 2010. Black – average score for each AGM course by the students. Red – average score for all AGM courses. Blue – average score for all SEP courses. Last three editions scored the highest. (Color figure online)

focused on the fact that given such a short space of time allocated in class, there was far too broad material in the Agile space to impart in further depth onto the students. In particular, we received comments such as:

"The course content is not enough to stretch 5 days."

"The large amounts of material made timing difficult."

This is a clear acknowledgment that the amount of material around Agile methods makes teaching the subject very difficult, especially in an intensive teaching environment. There was some skepticism about the game-like exercises, however, on further reflection with students several weeks after the course had completed even those students acknowledged that they had taken on board the learning from the exercises when they returned to their own organizations to share their new-found knowledge around Agile. Some of the positive comments we received included:

"Excellent way of teaching! You have expanded my horizon and gave me an excellent introduction to Agile."

"One of the best courses I have studied at Oxford. The lecturer and their use of guests made the course better and maybe of benefit to other courses! Initially I had doubts about the lecturer coming from industry but for this course it works better as they can draw on experience."

"A useful course with a timeliness of current industry trends."

"Good and helpful lecturers. The idea of students studying and presenting a case study is brilliant and helped a lot with understanding and discussing."

We have kept the Agile Methods course content timely by consciously putting Agile into practice in how we prepare and deliver the Agile Methods course itself. Feedback, in particular via the retrospective exercise, has allowed us to keep the course up to date with student and industry needs.

4 Related Work

Related work has mainly focused on teaching Agile to undergraduate and graduate students as part of computer science and software engineering curriculum's. The majority of these courses are typically group based projects, last 10–16 weeks, and teach Scrum and or XP. Our work reports on teaching a novel Agile methods course to *software engineering professionals* that are already working within industry and likely already have a degree, potentially in a computing subject, or have extensive software industry experience.

Lu and DeClue [12] discuss how Agile skills improve the marketability of new graduates. They also highlight the challenges posed in teaching Agile to undergraduates that stem from prerequisite experience and maturity. These challenges include fostering Agile approaches to skills such as communication, self-organization, and teamwork, where students who have less experience in a workplace may find mastery of these skills more difficult.

A panel at SIGCSE 2016 [3] raised a number of issues for teaching Agile methods in software engineering courses at a variety of computer science programs. The panel focused only on undergraduate university teaching (100 to 400 levels), hence novices to Agile with limited development experience.

Anslow et al. [1] reported their experience of teaching Agile methods to undergraduate and graduate students and presented a course outline along with associated teaching materials. They recommended not to teach the course to different levels simultaneously due to the nature of different levels of assessment required, abilities of the students, and additional administrative overheads.

Steghöfer et al. [16] reported on their efforts to improve teaching Agile, and Scrum in particular. They aimed to teach in a realistic manner but without encountering the technical difficulties of creating a real product by introducing exercises decoupled from software, such as LEGO Scrum.

Kropp et al. [10,11,13] looked at the status of Agile in education and industry and proposed a competency model on which to base integration of Agile into undergraduate teaching at two different universities. They found the most difficult competencies to teach are Agile values and management practices which they put significant emphasis on. Our AGM course also focuses on values and management practices and we have a complimentary course that focuses on Agile engineering practices[12] such as TDD and continuous integration.

Soundararajan, et al. [15] developed an advanced graduate-level course (to non-software professionals) in Agile software engineering at Virginia Tech. Their

[12] http://www.cs.ox.ac.uk/softeng/subjects/APE.html.

course has similarities to our approach where they focus on Agile product development, host guest talks from industry experts, and encourage students to present and debate Agile case studies within the class.

5 Conclusions

For today's computer science students who look towards entering a career in software engineering, skills beyond programming and technical excellence are essential. For any new graduate entering the tech industry, knowledge of Agile is essential. We hope that by sharing our extensive experiences in teaching Agile we can help foster excellence in Agile methods education in formal educational settings, such as in high school, university degree programs, and perhaps also in industrial training. From our experiences in teaching the Agile Methods course at the University of Oxford, we can extract many aspects of what we taught to graduate students that could be applied in any Agile teaching course. We believe that putting Agile theory into practice with a hands-on approach will lead to more effective learning. Based on the material reported in this paper other academics who wish to run similar courses can learn from our experiences.

6 Agile Methods: Case Study Papers for 2016

P1. M. Albisetti. Launchpad's quest for a better and agile user interface. In *XP*, pages 244–250. Springer, 2010.
P2. K. Boekhout. Mob programming: find fun faster. In *XP*, pages 185–192. Springer, 2016.
P3. C. Fry and S. Greene. Large scale agile transformation in an on-demand world. In *AGILE*, pages 136–142. IEEE, 2007.
P4. S. Hublikar and S. Hampiholi. Pause, reflect and act, the pursuit of continuous transformation. In *XP*, pages 201–208. Springer, 2016.
P5. M. Keeling. Put it to the test: Using lightweight experiments to improve team processes. In *XP*, pages 287–296. Springer, 2010.
P6. T. Little, F. Greene, T. Phillips, R. Pilger, and R. Poldervaart. Adaptive agility. In *AGILE*, pages 63–70. IEEE, 2004.
P7. S. McCalden, M. Tumilty, and D. Bustard. Smoothing the transition from agile software development to agile software maintenance. In *XP*, pages 209–216. Springer, 2016.
P8. B. Pieber, K. Ohler, and M. Ehegötz. University of Vienna's u:space turning around a failed large project by becoming agile. In *XP*, pages 217–225. Springer, 2016.
P9. D. Poon. A self funding agile transformation. In *AGILE*, pages 342–350. IEEE, 2006.
P10. M. Rajpal. Lessons learned from a failed attempt at distributed agile. In *XP*, pages 235–243. Springer, 2016.
P11. N. Robinson. A technical story. In *AGILE*, pages 339–343. IEEE, 2007.

P12. K.H. Rolland, V. Mikkelsen, and A. Næss. Tailoring agile in the large: Experience and reflections from a large-scale agile software development project. In *XP*, pages 244–251. Springer, 2016.

P13. C. Sudbery. How XP can improve the experiences of female software developers. In *XP*, pages 261–269. Springer, 2016.

P14. A. Takats and N. Brewer. Improving communication between customers and developers. In *AGILE*, pages 243–252. IEEE, 2005.

P15. I. Tsyganok. Pair-programming from a beginner's perspective. In *XP*, pages 270–277. Springer, 2016.

P16. B. Victor and N. Jacobson. We didn't quite get it. In *AGILE*, pages 271–274. IEEE, 2009.

Acknowledgments. Thanks to Jeremy Gibbons and Jim Davies from the Software Engineering Programme at the University of Oxford for their support. Thanks to guest lectures by Antony Marcano, Duncan Pierce, Lazaro Wolf, and Robert Biddle. Thanks to Rob Chatley for expert advice. Thanks to Clint Sieunarine and Ross Gales for being teaching assistants.

References

1. Anslow, C., Maurer, F.: An experience report at teaching a group based agile software development project course. In: SIGCSE, pp. 500–505. ACM (2015)
2. Beck, K., Andres, C.: Extreme Programming Explained: Embrace Change. Addison Wesley, Reading (2004)
3. Campbell, J., Kurkovsky, S., Liew, C.W., Tafliovich, A.: Scrum and agile methods in software engineering courses. In: SIGCSE, pp. 319–320. ACM (2016)
4. Cockburn, A.: Agile Software Development: The Cooperative Game. Addison Wesley, Boston (2006)
5. Derby, E., Larsen, D.: Agile Retrospectives: Making Good Teams Great. Pragmatic Bookshelf, Raleigh (2006)
6. Hazzan, O., Dubinsky, Y.: Why software engineering programs should teach agile software development. SIGSOFT Softw. Eng. Notes **32**(2), 1–3 (2007)
7. Lee, S.H., Lee, J., Liu, X., Bonk, C.J., Magjuka, R.J.: A review of case-based learning practices in an online MBA program: a program-level case study. Educ. Technol. Soc. **12**(3), 178–190 (2009)
8. Highsmith, J.: Agile Software Development Ecosystems. Addison Wesley, Boston (2002)
9. Kerth, N.L.: Project Retrospectives: A Handbook for Team Reviews. Dorset House Publishing Co., New York (2001)
10. Kropp, M., Meier, A.: Teaching agile software development at university level: Values, management, and craftsmanship. In: International Conference on Software Engineering Education and Training (CSEET), pp. 179–188. IEEE (2013)
11. Kropp, M., Meier, A.: New sustainable teaching approaches in software engineering education. In: EDUCON, pp. 1019–1022. IEEE (2014)
12. Lu, B., DeClue, T.: Teaching agile methodology in a software engineering capstone course. J. Comput. Sci. Coll. **26**(5), 293–299 (2011)

13. Meier, A., Kropp, M., Perellano, G.: Experience report of teaching agile collaboration and values: agile software development in large student teams. In: International Conference on Software Engineering Education and Training (CSEET), pp. 76–80. IEEE (2016)
14. Schwaber, K., Beedle, M.: Agile Software Development with Scrum. Pearson, Upper Saddle River (2001)
15. Soundararajan, S., Chigani, A., Arthur, J.D.: Understanding the tenets of agile software engineering: Lecturing, exploration and critical thinking. In: SIGCSE, pp. 313–318. ACM (2012)
16. Vogel, B., Kilamo, T., Kurti, A.: Teaching distributed agile development to software professionals: a flexible approach. In: European Conference on Software Architecture Workshops, ECSAW, pp. 31:1–31:8. ACM (2015)