

Bobby Tait
Jan Kroeze
Stefan Gruner (Eds.)

Communications in Computer and Information Science

1136

ICT Education

48th Annual Conference of the Southern African
Computer Lecturers' Association, SACLA 2019
Northern Drakensberg, South Africa, July 15–17, 2019
Revised Selected Papers

Communications in Computer and Information Science


1136

Commenced Publication in 2007

Founding and Former Series Editors:

Phoebe Chen, Alfredo Cuzzocrea, Xiaoyong Du, Orhun Kara, Ting Liu,
Krishna M. Sivalingam, Dominik Ślęzak, Takashi Washio, Xiaokang Yang,
and Junsong Yuan

Editorial Board Members

Simone Diniz Junqueira Barbosa 

*Pontifical Catholic University of Rio de Janeiro (PUC-Rio),
Rio de Janeiro, Brazil*

Joaquim Filipe 

Polytechnic Institute of Setúbal, Setúbal, Portugal

Ashish Ghosh

Indian Statistical Institute, Kolkata, India

Igor Kotenko 

*St. Petersburg Institute for Informatics and Automation of the Russian
Academy of Sciences, St. Petersburg, Russia*

Lizhu Zhou

Tsinghua University, Beijing, China

More information about this series at <http://www.springer.com/series/7899>


Bobby Tait · Jan Kroeze ·
Stefan Gruner (Eds.)

ICT Education

48th Annual Conference of the Southern African
Computer Lecturers' Association, SACLA 2019
Northern Drakensberg, South Africa, July 15–17, 2019
Revised Selected Papers

Editors

Bobby Tait
School of Computing
University of South Africa
Johannesburg, South Africa

Jan Kroeze 
School of Computing
University of South Africa
Johannesburg, South Africa

Stefan Gruner
Department of Computer Science
University of Pretoria
Pretoria, Gauteng, South Africa

ISSN 1865-0929 ISSN 1865-0937 (electronic)
Communications in Computer and Information Science
ISBN 978-3-030-35628-6 ISBN 978-3-030-35629-3 (eBook)
<https://doi.org/10.1007/978-3-030-35629-3>

© Springer Nature Switzerland AG 2020

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, expressed 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 Switzerland AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

Preface

This volume of CCIS contains the revised selected papers of SACLA 2019, the 48th Annual Conference of the Southern African Computer Lecturers' Association,¹ held at Alpine Heath in the Northern Drakensberg region, South Africa,² July 15–17, 2019, whereby it is now for the 4th time that our revised post-proceedings are published by Springer-Verlag in this CCIS series.³ The Northern Drakensberg region is well known for many famous natural features, including the Amphitheatre, which rises a thousand metres straight up over a distance of five kilometres, and the Tugela Falls, which is the second-highest waterfall in the world.

The objective of SACLA is to promote cooperation among South African, southern African, as well as international experts, specifically to further the method and contents of university education in the field of computing. The annual conference brings together lecturers, researchers, scientists, post-graduate students, industry, and society stakeholders interested in the field of computing education, and for this reason it provides opportunities for networking and for learning from one another.

The theme of this conference was: “Computing Matters of Course”. Our conference logo not only points at what makes a successful lecturer but also at the role SACLA 2019 would play in getting our minds and heads in gear. Intending to help others to become better computing lecturers, SACLA 2019 invited national and international submissions that focused on practical experiences and successes in computing education at the tertiary level, in the following topic areas:

- Classroom innovation, and assessment of the impact thereof
- Novel tools developed, or novel use of existing tools, for purposes of learning and/or assessment
- Research undertaken to investigate aspects of computing education

Our Program Committee had more than 50 members, whereby more than half of them were international members (from outside southern Africa). This international mixture of the Program Committee ensures that all papers presented at our conference meet international quality standards. Each submitted paper was reviewed by three members of the Program Committee in a rigorous, double-blind mode, whereby especially the following criteria were taken into consideration: topic choice, problem choice, helpfulness for the ICT lecturer's self-improvement, coverage of related work and accuracy of references, empirical findings, solution development, as well as language and technical editing. Of the three reviews for each submission, at least one was provided by an international reviewer (from outside southern Africa).

¹ <http://www.sacla.org.za/index.php/about/>

² <http://sacla2019.unisa.ac.za/>

³ Previous volumes: CCIS **642**, **730**, **963**.

After the international dissemination of our call for papers, 57 submissions were initially received and carefully reviewed. 25 of them were accepted for presentation at the SACLA 2019 conference as full papers, 6 as short papers, and 12 as extended abstracts. 31 of those were actually presented. Of those full papers, a fine selection of 16 revised and further improved versions were finally accepted for inclusion in this volume of CCIS as the conference's most noteworthy contributions. The overall paper acceptance rate for this CCIS book is thus 28%, which shows our commitment to high academic quality.

Thereby, SACLA's ongoing process of internationalization, which was started at SACLA 2016 is making progress and already bears fruit: the papers compiled in this volume of CCIS have authors or co-authors from universities located (in alphabetical order) in Germany, Kenya, the Netherlands, Norway, Puerto Rico, and South Africa. We hope to be able to continue to attract international submissions also in the forthcoming years, because a globalised world requires the exchange of good ideas from everywhere.

We were fortunate to have Dolf Steyn (from South Africa) as the keynote speaker. The topic of his interesting motivational talk was: "Is Big Data a DNA or a Diet Problem? Either way, it does not fit comfortably yet."⁴ Moreover, the paper by Pakiso J. Khomokhoana and Liezel Nel, "Decoding Source Code Comprehension: Bottlenecks Experienced by Senior Computer Science Students",⁵ received the conference's Best Paper Award.

Affiliated with our conference were the following three successful workshops:

- The South African Computing Accreditation Board (SACAB) Meeting, hosted by André Calitz as facilitator
- Academic Writing for Junior Informaticians and Computer Scientists, hosted by Stefan Gruner as facilitator and instructor
- Amazon Web Services (AWS) Educate and Academy Programs, hosted by Amazon's South African branch

Moreover, our conference also had two interesting Open Discussion Forums on the following topics

- The Corporatisation of Universities, hosted by Colin Pilkington and Wynand van Staden as moderators
- Perspectives for the Establishment of Didactics of Informatics Chairs at South African Universities, hosted by Stefan Gruner as moderator

We hope that the ideas exchanged at these affiliated workshops and open discussion forums will continue to grow and bear fruit in the future.

We extend our thanks and appreciation to the conference's general chair, Mac van der Merwe, the entire Organising Committee, and all colleagues and friends who contributed to the success of SACLA 2019. On behalf of the SACLA community we also express our deepest appreciation to our sponsors: Oracle Academy, AWS Educate,

⁴ Not included as a paper in this volume.

⁵ Paper #2, on page 17 of this volume.

IITPSA, Cambridge University Press, Smart SWOT, and Cengage Learning, without whom our conference at Alpine Heath would not have been possible. Also the friendly and helpful service personnel of Alpine Heath shall not be forgotten. Further words of thanks are addressed to the authors for having chosen SACLA 2019 as the forum for communicating their noteworthy insights and interesting thoughts. Many thanks also to the members of our international Program Committee, who—also with help from several additional reviewers—all provided extensive and insightful reviews. Last but not least, many thanks also to the helpful staff of Springer-Verlag, who made this CCIS publication possible.

Throughout the remainder of this book, the term ‘ICT’ stands for information and communication technologies, comprising computer science, information systems, information science, and related areas of studies (which cannot be sharply distinguished from each other). We wish our readers a fruitful reading experience with this volume of CCIS, and we look forward to the continuation of the SACLA series in the following years.



The sponsors and supporters of SACLA 2019 are herewith gratefully acknowledged

October 2019

Bobby L. Tait
Jan H. Kroeze
Stefan Gruner

Organization



General Chair

Mac van der Merwe UNISA Johannesburg, South Africa

Programme Committee Co-Chairs

Bobby Tait UNISA Johannesburg, South Africa

Jan Kroeze UNISA Johannesburg, South Africa

Local Arrangements and Additional Support

Mathias Mujinga UNISA Johannesburg, South Africa

Hanifah Abdullah UNISA Johannesburg, South Africa

Bester Chimbo UNISA Johannesburg, South Africa

Colin Pilkington UNISA Johannesburg, South Africa

SACLA Publications Chair and CCIS Post-Proceedings Co-Editor

Stefan Gruner University of Pretoria, South Africa

Programme Committee

Juan-Carlos Augusto Middlesex University, London, UK

Maurice ter Beek ISTI-CNR, Pisa, Italy

Dines Bjørner Lyngby Technical University of Denmark, Denmark

Jürgen Börstler Blekinge Institute of Technology, Sweden

Mark van den Brand* Technische Universiteit Eindhoven, The Netherlands

Torsten Brinda* Universität Duisburg-Essen, Germany

Helmut Caba Pädagogische Hochschule Salzburg, Austria

Serge Chaumette LaBRI/Université de Bordeaux, France

Loek Cleophas+ Technische Universiteit Eindhoven, The Netherlands

Andrea Corradini Università di Pisa, Italy

Marian Daun Universität Duisburg-Essen, Germany

Jörg Desel Fernuniversität in Hagen, Germany

Gordana Dodig-Crnkovic Chalmers University of Technology, Göteborg,
Sweden

Rachid Echahed	Université de Grenoble, France
Marko van Eekelen*	Radboud Universiteit, Nijmegen, The Netherlands
Sigrid Ewert	University of the Witwatersrand, Johannesburg, South Africa
Carla Ferreira	Universidade Nova de Lisboa, Portugal
Karl-Josef Fuchs	Universität Salzburg, Austria
Kurt Geihs*	Universität Kassel, Germany
Jaco Geldenhuys	Stellenbosch University, South Africa
Richard Glassey	Robert Gordon University, Aberdeen, UK
Stefan Gruner*	University of Pretoria, South Africa
Jurriaan Hage	Universiteit Utrecht, The Netherlands
Pieter Hartel	Universiteit Twente, The Netherlands
Michaela Huhn	Ostfalia Hochschule für angewandte Wissenschaften, Wolfenbüttel, Germany
Fourie Joubert	Universiteit van Pretoria, South Africa
Agnes Koschmider	Karlsruher Institut für Technologie, Germany
Eduan Kotzé*	University of the Free State, South Africa
Hans-Jörg Kreowski	Universität Bremen, Germany
Jan Kroeze	UNISA, Johannesburg, South Africa
Nguyen-Thinh Le	Humboldt-Universität zu Berlin, Germany
Horst Lichter*	RWTH, Aachen, Germany
Reza Malekian	Malmö University, Sweden
Carlos Matos	Royal Holloway University of London, UK
Greg Michaelson	Heriot-Watt University, Edinburgh, UK
Thomas Noll	RWTH, Aachen, Germany
Peter-Csaba Ölveczky	University of Oslo, Norway
Barbara Paech	Universität Heidelberg, Germany
Nelishia Pillay	University of Pretoria, South Africa
Karen Renaud*	Abertay University, Dundee, UK
Arend Rensink	Universiteit Twente, The Netherlands
Ian Sanders*	UNISA, Johannesburg, South Africa
Holger Schlingloff	Humboldt-Universität zu Berlin, Germany
Gerardo Schneider	Chalmers University of Technology, Göteborg, Sweden
Monika Seisenberger	Swansea University, UK
Bobby Tait	UNISA, Johannesburg, South Africa
Bob Travica	University of Manitoba, Winnipeg, Canada
Lorna Uden	Staffordshire University, Stoke-on-Trent, UK
Janis Voigtländer	Universität Duisburg-Essen, Germany
George Wells	Rhodes University, Grahamstown, South Africa
Bernhard Westfechtel	Universität Bayreuth, Germany
Uwe Wolter	University of Bergen, Norway
Akka Zemhari	LaBRI/Université de Bordeaux, France
Olaf Zukunft*	Hochschule für angewandte Wissenschaften, Hamburg, Germany

Additional Reviewers

Alexander, Peter
Babur, Önder
Denisova, Alena
Fleitas, Yeray del C. B.
Hacks, Simon
Jahl, Alexander
James, Phillip
Karkhanis, Priyanka
Kuiper, Ruurd

Ossenkopf, Marie
Plewnia, Christian
Rohmann, Astrid
Rauchas, Sarah
Safari, Solmaz
van Staden, Wynand
Staudemeyer, Ralf C.
Walton, Sean

PC members marked with * are continuing PC members from the previous year's conference, SACLA 2018 (Springer: CCIS **963**), and PC members marked with + had acted as additional reviewers for SACLA 2018.

Contents

Computer Programming Education

- Synthesis of Social Media Messages and Tweets as Feedback Medium
in Introductory Programming 3
Sonny Kabaso and Abejide Ade-Ibijola
- Decoding Source Code Comprehension: Bottlenecks Experienced
by Senior Computer Science Students 17
Pakiso J. Khomokhoana and Liezel Nel

System Security Education

- An Approach to Teaching Secure Programming in the .NET Environment . . . 35
Sifiso Bangani, Lynn Futcher, and Johan van Niekerk
- A Framework for Integrating Secure Coding Principles into
Undergraduate Programming Curricula. 50
Sandile Ngwenya and Lynn Futcher
- Developing a Digital Forensics Curriculum: Exploring Trends
from 2007 to 2017 64
Roshan Harneker and Adrie Stander

Software Engineering Education

- Hackathons as a Formal Teaching Approach in Information Systems
Capstone Courses 79
Walter F. Uys
- Modernizing the Introduction to Software Engineering Course 96
Marko Schütz-Schmuck
- Exercise Task Generation for UML Class/Object Diagrams,
via Alloy Model Instance Finding 112
Violet Kafa, Marcellus Siegburg, and Janis Voigtländer

Education of Post-Graduate Research-Students

- A Connectivist View of a Research Methodology Semantic Wiki 131
Colin Pilkington and Laurette Pretorius

Cohort Supervision: Towards a Sustainable Model for Distance Learning. . . . 147
Judy van Biljon, Colin Pilkington, and Ronell van der Merwe

Guidelines for Conducting Design Science Research
in Information Systems 163
Alta van der Merwe, Auroa Gerber, and Hanlie Smuts

Our Students, Our Profession

Making Sense of Unstructured Data: An Experiential Learning Approach . . . 181
Sunet Eybers and Marie J. Hattingh

Connecting Generation Z Information Systems Students to Technology
Through the Task-Technology Fit Theory 197
*Adriana A. Steyn, Carina de Villiers, Joyce Jordaan,
and Tshегоfatso Pitso*

Detecting Similarity in Multi-procedure Student Programs Using only
Static Code Structure. 211
Karen Bradshaw and Vongai Chindeka

Enhancing Computer Students' Academic Performance Through
Explanatory Modeling 227
Leah Mutanu and Philip Machoka

The Use of Industry Advisory Boards at Higher Education Institutions
in Southern Africa. 244
Estelle Taylor and Andre P. Calitz

Author Index 261

Computer Programming Education



Synthesis of Social Media Messages and Tweets as Feedback Medium in Introductory Programming

Sonny Kabaso  and Abejide Ade-Ibijola  

Department of Applied Information Systems, University of Johannesburg,
Johannesburg, South Africa
abejideai@uj.ac.za

Abstract. Social Media have been recognised as supportive tools in education for creating benefits that supplement students' collaboration, class interactions, as well as communication between instructors and students. Active informal interaction and feedback between instructors and students outside class belong to the main reasons behind social media pedagogy. Despite the prevalence of traditional email methods of providing feedback to students, the literature shows that they do not check their emails as frequently as they check their social media accounts. In this paper we present the automatic generation of feedback messages and tweets with context-free grammars (CFG). Our system takes a class list of students and their mark sheets and automatically composes Twitter tweets concerning statistical 'fun facts' about programming problems, exercises, class performances, as well as private messages about individual student performances. A survey with 116 participating students showed that the majority of them would like to receive such notifications on social media rather than emails. Lecturers found our system promising, too.

Keywords: Introductory programming · Social media · Tweet synthesis · Context-free grammar · Procedural generation

1 Introduction

Social media are a collection of web-based applications built on the foundations of Web 2.0 technology. They consist of a multitude of user-generated and shared information [19]. Over the years social media have come to count to the most popular internet services in the world [14]. They made remote communication possible for many people [25], and became an essential part of our everyday lives [18]. Despite the fact that social media were only used for social communication when first introduced, they can be used in professional contexts and have

This work is financially supported by the National Research Foundation (NRF) of South Africa (Grant No. 119041). Bridge Labs Inc., Johannesburg, provided an additional travel grant.

evolved into a platform for student engagement and interaction [8,37], including collaboration and feedback [1,10]. For instance, **Twitter** has driven instant interaction between instructors and their students, student participation, formal communication outside the classroom, and continued learning when students are off-campus [16,20,26]. ‘Digital natives’ are frequent social media users and would rather receive information using modern technology; this aids traditional learning and teaching methods [10,13,17].

Some studies have shown that when social platforms like **Twitter** are used in an educational setting, learning is more engaging and stimulating than traditional teaching methods [26,39]. Students also prefer to use mobile devices for social media because they have access to their devices at any time, thus extending the time of interaction outside the classroom [8,10,34]. Thus, incorporating social media in education can improve the amount of time students interact with their studies. This should also be beneficial for new students in introductory programming courses.

Introducing new students to computer programming requires time and patience [15,22]. It is considered a challenge by many computer programming instructors [32]. Many novice programmers find—and have always found—learning new programming concepts difficult [12], with many publications providing evidence of high failure and drop-out rates [7,15,23,33,38]. Novice programmers have minimal engagement with programming concepts and problems. They fail to create clear mental models of programme execution [27]. In order to address this problem, we need to develop teaching methods that enable regular practice by programming students, preferably using social media as auxiliary tools [24]. Given that many institutions still depend on traditional pedagogical models of teaching in our digital era, the amount of time students have with ‘classical’ course content is rather limited as they are almost always ‘on’ social media [37]. Hence, it has become imperative that we create innovative methods of engaging with the ‘digital natives’ also in the programming courses.

One innovation will be to send ‘push’ notifications to novice programmers on social media, automatically. With social media’s popularity, creating supplementary communication methods among students and instructors can have positive learning effects [34]. Therefore, in this paper, we present the synthesis of social media messages and ‘tweets’ as feedback means for novice programmers by employing techniques from natural language generation (NLG) [30,31] and formal (context-free) grammars. We generate tweets, taking into account student information and predefined templates. Our tweets are categorised into ‘broadcasts’ (for everybody) or direct messages (to individual students). Broadcasts include performance statistics, learning guides, test announcements, exercises, assignment notifications, as well as new topics and concepts. Direct messages refer to individual students’ attendance and performance. This communication processes is depicted in Fig. 1.

In the process shown in Fig. 1, an instructor inserts a class list containing students’ course information into a database. The **Tweet Synthesizer** then automatically generates feedback tweets and messages. These tweets and messages are pushed to **Twitter** via its web API (Appl. Progr. Interface).

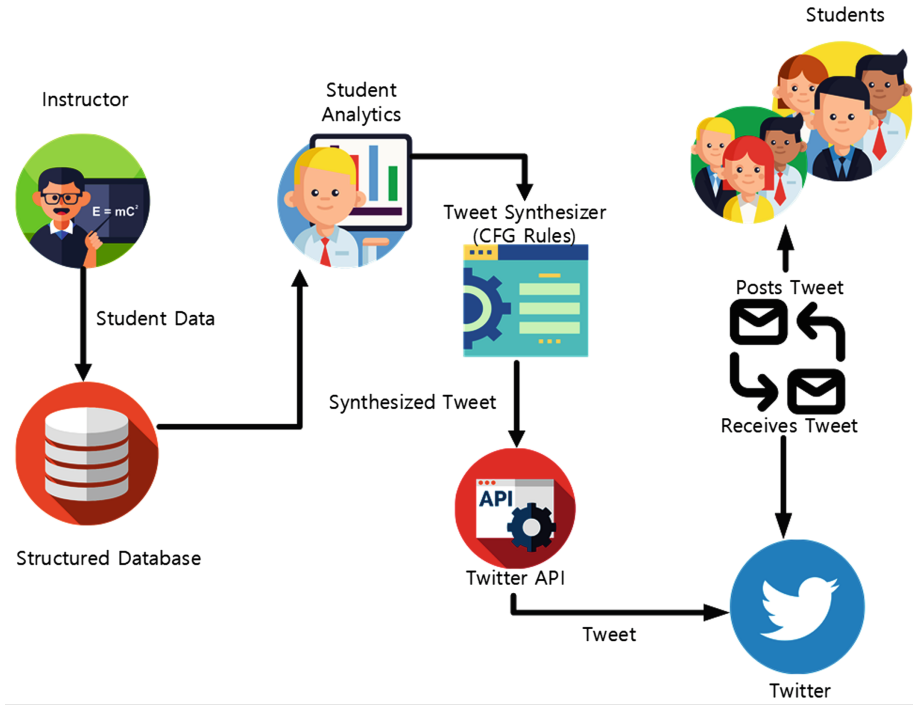


Fig. 1. Process of tweet feedback generation with CFGs

In the following sub-sections we define somewhat more precisely what was the motivation for our work and what research questions we wanted to address. For the sake of clarity we also recapitulate the well-known definition of context-free grammars. Thereafter, the rest of this paper is organised as follows: Sect. 2 outlines some related work. Section 3 describes our CFG for tweet generation. Section 4 shows the implementation of our tool together with examples of generated tweets. Section 5 evaluates and discusses our achievement. Section 6 concludes the paper and hints at further work to be done.

1.1 Motivation

Our work is motivated by the following problems faced by programming instructors and novice students.

Location and Time Constraints: Time and location constraints become a major setback in a traditional setting because students only interact with course work during ‘school time’ [10].

Students’ Preference: Students use social media more compared to e-mail [10, 17]. Also [34] indicated that students preferred to use the social media rather than e-mail.

Course Difficulty: Programming is a difficult subject to learn [7, 12, 15, 23, 38].

In order to obtain long-term knowledge and proficiency it is important that novice programmers have continuous interaction and practice with programming puzzles [24].

1.2 Research Questions

Thus motivated, this paper shall answer the following research questions:

1. How can we generate personalised social media messages and tweets as feedback, based on real-time data on students' attendance and performance, with limited or no lecturer involvement?
2. How can we improve learning methods for Introductory Programming and make it more interactive outside normal classes in informal contexts?
3. How can we define CFG rules to implement such a system?

1.3 Contribution

For the purpose of answering our research questions we have:

- developed a method of automatically generating educational feedback to support Introductory Programming in the form of tweets using CFG rules;
- built a tool called the **Tweet Synthesiser** that automatically generates and posts programming concepts, practice exercises and solutions in the form of tweets for novice programmers;
- evaluated how useful the **Tweet Synthesiser** is for novice programmers.

1.4 Context-Free Grammars

A concept that appears many times in this paper is that of a context-free grammar. From [5] we adopt the following

Definition: Context-Free Grammar (CFG). A CFG is a quadruple $G = (N, \Sigma, P, S)$, where:

1. N is a restricted set of elements known as non-terminal or syntactic variables.
2. Σ is a restricted alphabet of terminal symbols where $\Sigma \not\subseteq N$. Terminals appear on the right side of Replacing Rules
3. P is a set of Replacing Rules, with the form $A \longrightarrow a$, where $A \in N$ and $a \in (N \cup \Sigma)^*$.
4. S is the selected starting non-terminal where $S \in N$.

2 Related Work

Twitter is a popular micro-blogging platform for social networking which was established in 2006. It is estimated to have ≈ 555 million active users around the world [28]. Many of them are ‘digital natives’ and students [8]. It allows for people to communicate using short text messages (280 characters limit) or status updates known as ‘tweets’. These can be posted via the Twitter application, instant text messaging via other third party applications (e.g. Facebook), e-mail, or web sites using the Twitter API [36]. Twitter is easily accessible regardless of the geographical location of the user.

Twitter being a ‘push’ technology, feedback is instant because the Twitter mobile application is readily available on students’ mobile devices [10] as soon as it is downloaded and installed. In the education domain, Buettner has described how to ‘push’ concepts, subject-related resources and new topics to Twitter with the aim of each student having direct notifications on a mobile device every morning. The majority of the students in that study found this practice much more engaging and useful [9, 39]. Although there has not been any work directly related to automatically generating messages and tweets as feedback to Twitter, previous works in NLG and Problem Generation are mentioned as follows.

Natural Language Generation. Generation of social media profiles using probabilistic CFGs on Facebook can be found in [4]. Human-like language generation using Monte Carlo Tree Search employing context-free grammars is described in [21]. Sentence generation for probabilistic TAG grammars can be found in [6, 11]. The generation of narratives of SQL queries using context-free grammars is described in [29], and in-game text generation with expressive free text markup and CFGs in [35].

Problem Generation. Automatic generation of python practice problems using context-free grammars can be found in [2], and automatic generation of regular expression practice problems (with solutions) in [3].

3 Grammar Design for Tweet Synthesis

3.1 Structure of a Tweet

Before we can design the grammar for synthesising tweets, we first need to outline the structure and components of a tweet. A tweet can contain up to 280 characters, including spaces [40]. In this paper we assume a tweet to be made of four components:

1. **text:** the short *message* that a user wants to deliver to a specified audience;
2. **hashtag:** a word or phrase preceded by the hash symbol #. This is used to indicate (as ‘meta data’ for purposes of search and retrieval) a topic, event, or community associated to the tweet. A tweet can contain one or more hashtags inclusive to the allowed 280 characters;

3. **mention:** the *user name* or Twitter ‘handle’ of a specific user, preceded by the ‘at’ symbol @;
4. **url:** a web link to an external source outside the Twitter domain. It is usually related to the *context* of the tweet.

Each of these four components is optional though at least one of them must occur in every tweet, (i.e.: no empty tweets).

3.2 Building Blocks

Rule (1) defines an `<identifier>` via the regular expression $[A - Za - z0 - 9_]^+$. Rule (2) states that a ‘mention’ must be preceded by the ‘@’ symbol representing a Twitter handle or user name. Rule (3) is similar whereby a `<hashtag>` must always be preceded by #. Rule (4) defines the context of the message being tweeted. Rule (5) define the message itself. Rule (7) specifies any permissible URL with less than 280 characters.

$$\langle \text{identifier} \rangle \rightarrow [A - Za - z0 - 9_]^+ \quad (1)$$

$$\langle \text{mention} \rangle \rightarrow @\langle \text{identifier} \rangle \quad (2)$$

$$\langle \text{hashtag} \rangle \rightarrow \#\langle \text{identifier} \rangle \quad (3)$$

$$\langle \text{context} \rangle \rightarrow \langle \text{exercise} \rangle | \langle \text{schedule} \rangle | \langle \text{attendance} \rangle | \langle \text{performance} \rangle \quad (4)$$

$$\langle \text{msg} \rangle \rightarrow \langle \text{context} \rangle \langle \text{url} \rangle | \langle \text{context} \rangle \quad (5)$$

$$\langle \text{optional_txt} \rangle \rightarrow \langle \text{text} \rangle | \varepsilon \quad (6)$$

$$\langle \text{url} \rangle \rightarrow \textit{usual web address string of maximally 280 characters} \quad (7)$$

3.3 Tweet

Rule (8) states that a tweet can either be a broadcast `<broadcast_tweet>` or a direct message `<inbox_tweet>`:

$$\langle \text{tweet} \rangle \rightarrow \langle \text{broadcast_tweet} \rangle | \langle \text{inbox_tweet} \rangle \quad (8)$$

whereby the entire tweet must not exceed 280 characters

3.4 Broadcast Tweet

Rule (9) makes a sequence of hashtags optional. Rule (10) defines a broadcast. Contrary to the general rules of Twitter (see Subsect. 3.1), we here demand for our application that a tweet’s `<msg>` component *cannot* be optional. As usual, the entire tweet with all components may not exceed 280 characters.

$$\langle \text{tag} \rangle \rightarrow [\langle \text{hashtag} \rangle]^* \quad (9)$$

$$\begin{aligned} \langle \text{broadcast_tweet} \rangle \rightarrow & \langle \text{msg} \rangle | \langle \text{msg} \rangle \langle \text{tag} \rangle | \\ & (\langle \text{mention} \rangle \langle \text{msg} \rangle | \langle \text{msg} \rangle \langle \text{mention} \rangle) \langle \text{tag} \rangle | \\ & \langle \text{tag} \rangle (\langle \text{mention} \rangle \langle \text{msg} \rangle | \langle \text{msg} \rangle \langle \text{mention} \rangle) | \\ & \langle \text{msg} \rangle (\langle \text{mention} \rangle \langle \text{tag} \rangle | \langle \text{tag} \rangle \langle \text{mention} \rangle) | \\ & \langle \text{mention} \rangle (\langle \text{msg} \rangle \langle \text{tag} \rangle | \langle \text{tag} \rangle \langle \text{msg} \rangle) \end{aligned} \quad (10)$$

To produce a `<broadcast_tweet>` for an exercise or a solution we can further decompose the variables for `<exercise>` and `<solution>` (Rules 11–12), whereby `<optional_txt>` may contain any wording that ‘makes sense’ in the context of the tweet. Rule (13) defines specifies schedules of events.

$$\begin{aligned} \langle \text{exercise} \rangle &\rightarrow \langle \text{optional_txt} \rangle \langle \text{category} \rangle \langle \text{topic} \rangle \langle \text{optional_txt} \rangle \\ &\quad \langle \text{problem_no} \rangle \langle \text{problem_txt} \rangle \langle \text{optional_txt} \rangle \end{aligned} \quad (11)$$

$$\begin{aligned} \langle \text{solution} \rangle &\rightarrow \langle \text{optional_txt} \rangle \langle \text{category} \rangle \langle \text{topic} \rangle \langle \text{optional_txt} \rangle \\ &\quad \langle \text{solution_no} \rangle \langle \text{solution_txt} \rangle \langle \text{optional_txt} \rangle \end{aligned} \quad (12)$$

$$\begin{aligned} \langle \text{schedule} \rangle &\rightarrow (\langle \text{event} \rangle \langle \text{optional_txt} \rangle) | \\ &\quad (\langle \text{optional_txt} \rangle \langle \text{event} \rangle) \langle \text{schedule} \rangle \end{aligned} \quad (13)$$

3.5 Inbox Tweet

Finally we define the rules for producing direct inbox message tweets:

$$\langle \text{inbox_tweet} \rangle \rightarrow \langle \text{msg} \rangle \langle \text{tag} \rangle | \langle \text{tag} \rangle \langle \text{msg} \rangle \quad (14)$$

$$\begin{aligned} \langle \text{performance} \rangle &\rightarrow \langle \text{optional_txt} \rangle \langle \text{marks} \rangle \langle \text{module} \rangle \langle \text{optional_txt} \rangle \\ &\quad \langle \text{event} \rangle \end{aligned} \quad (15)$$

$$\langle \text{attendance} \rangle \rightarrow \langle \text{optional_txt} \rangle \langle \text{avg_attendance} \rangle \langle \text{optional_txt} \rangle \quad (16)$$

Note: unlike broadcast tweets, which are limited to 280 characters, *inbox tweets* are allowed to consist of more than 280 characters. Two examples of tweet productions from the grammar of above are given in the following paragraphs.

Example 1: Direct Message. In this example we produce a direct message about a student’s performance in some test. Rule (14) is the starting point.

```

<inbox_tweet>
→ <performance><hashtag>
→ <optional_txt><marks><module><optional_txt><event><hashtag>
→ You scored <marks><module><optional_txt><event><hashtag>
→ You scored 50% <module><optional_txt><event><hashtag>
→ You scored 50% DSW1A <optional_txt><event><hashtag>
→ You scored 50% DSW1A in <event><hashtag>
→ You scored 50% DSW1A in TEST1 28/06/2019 <hashtag>
→ You scored 50% DSW1A in TEST1 28/06/2019 #assessments

```

Example 2: Schedule Tweet. Here we produce a broadcast announcement for a scheduled test. Rule (10) is the starting point.

```

<broadcast_tweet>
→ <msg><hashtag>
→ <optional_txt><event><optional_txt><hashtag>
→ You have <event><optional_txt><hashtag>
→ You have Test1 on 28/06/2019. <optional_txt><hashtag>
→ You have Test1 on 28/06/2019. Do not forget to add it to your calendar.
  <hashtag>
→ You have Test1 on 28/06/2019. Do not forget to add it to your calendar.
  #assessments

```

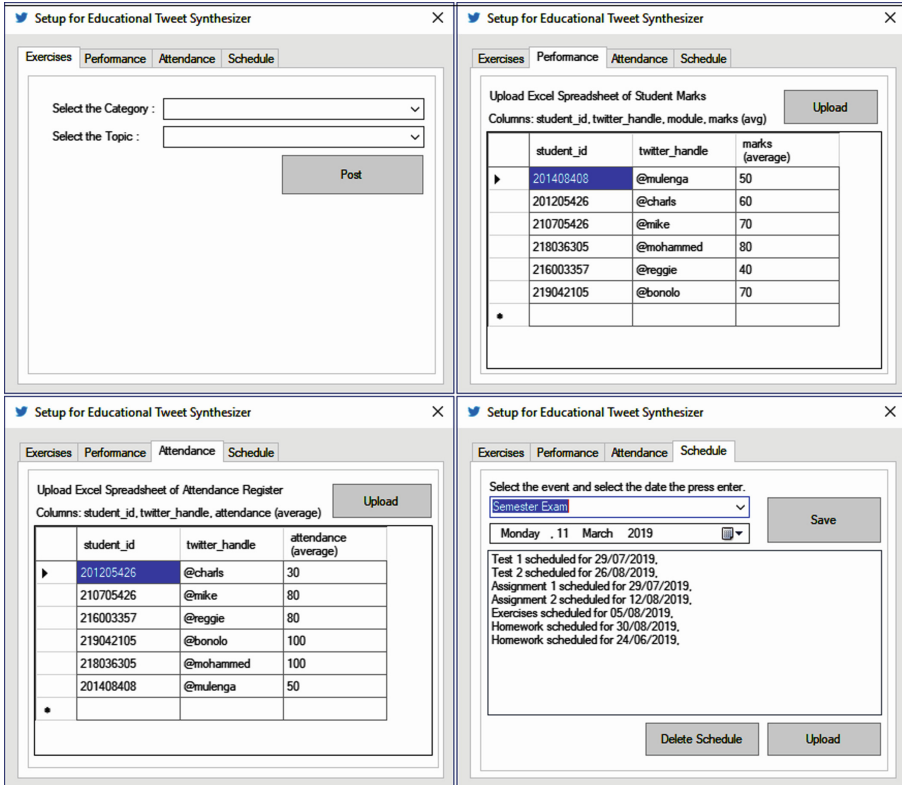


Fig. 2. Setup for educational **Tweet Synthesiser**: (a) Topic selection for exercise and solution, (b) Students' performance, (c) Attendance, (d) Schedule and deadlines

4 Implementation

In the previous section we presented our rules for generating tweets. Now we describe how we have implemented these rules in a push notification tool for messages and feedback on Twitter. Our tool, developed in the .Net framework, is shown in Fig. 2. It is divided into four sections for four tweet categories namely: *exercises/solutions*, *student performance*, *attendance*, and *schedule*. For an exercise tweet, we first select a programming category and a topic for the desired tweet. For instance, the category could be *algorithm design* and the topic *if-statements*. With those selections a problem exercise (with solution) is automatically generated via the AAI API [2]. We generate procedural practice algorithms and programs in *Python*.

The problem and its solution are then passed through the tweet synthesizer to post a broadcast exercise tweet with problem and solution temporally separated. Figure 4 shows two generated tweets for a problem exercise (a) and its solution (b). Our **Tweet Synthesiser** generated 500 tweet exercises and 500

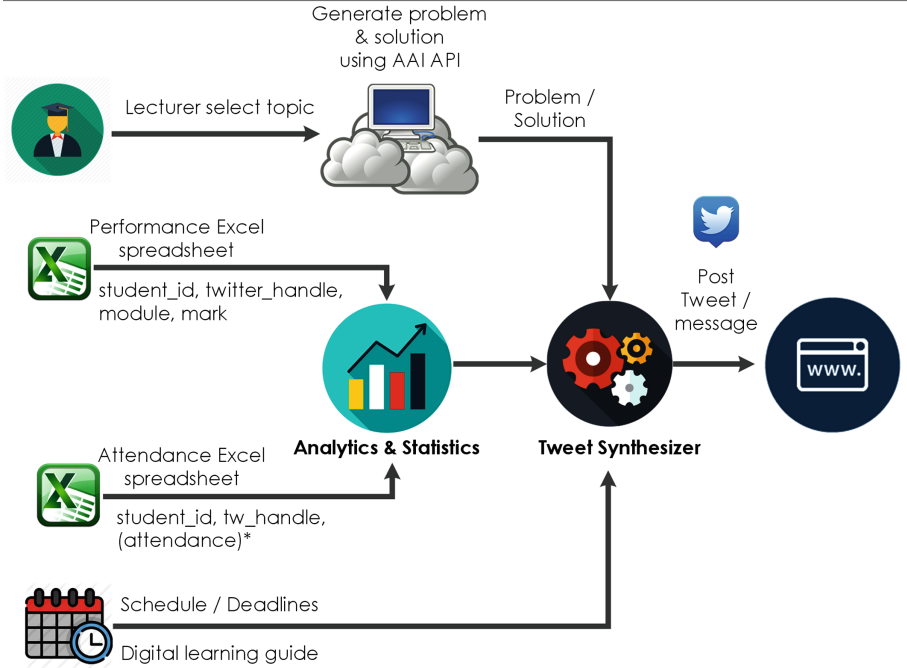


Fig. 3. Tweet Synthesiser: work-flow

tweet solutions for their corresponding questions within a minute of time. A five second interval separates the posting of two tweets to avoid ‘flooding’ the API.

Performance and attendance tweets work similarly: we select a class list with the fields (`student_id`, `twitter_handle`, `module` and `marks/attendance`) as a CSV file. With this information, we compose custom tweet messages depending on students’ performance average and class attendance; they are sent specifically to individual students. We also generate schedule or deadline tweets from a digital study-guide by inserting important events or dates. Figure 3 shows the flow of tweet generation with our tool. Some output examples are shown in Fig. 4.

5 Evaluation: Students’ Perception of Tweet Synthesiser

Next we describe the results of a survey concerning students’ opinions about receiving educational announcements via our **Tweet Synthesiser**. The survey was conducted online at our university. The majority of the respondents were enrolled in Information Technology and/or Information Systems curricula in which they take computer programming courses. 116 responses were received. About 92% of the respondents were students between the age of 18 to 25, 6.2% were above the age of 25, while the remaining 1.8% were less than 18 years of age, (Fig. 6a). Students were asked if their instructors used social media to support teaching and learning: only 17.7% answered *Yes* (thereby also considering

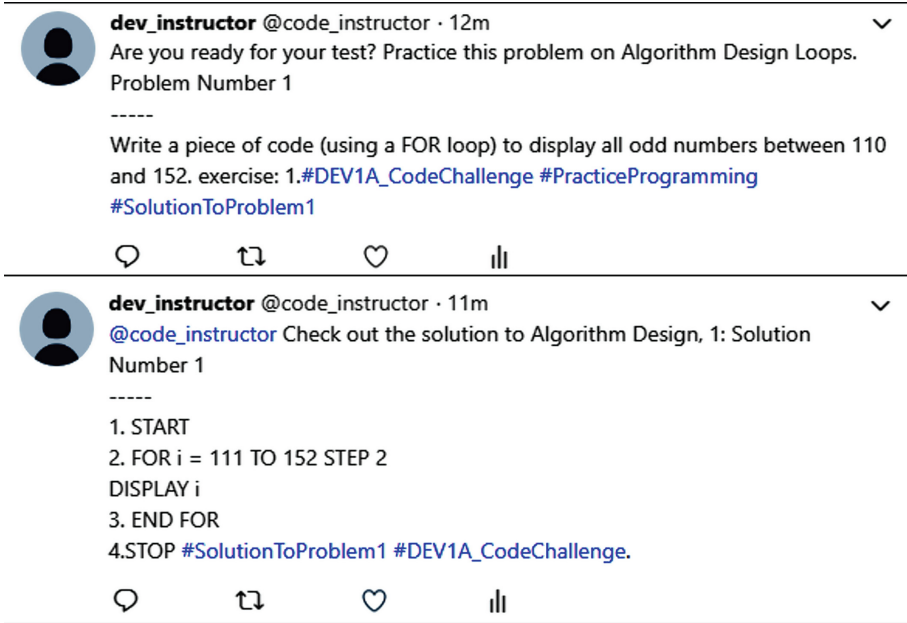


Fig. 4. Output tweets: (a) Problem task, (b) Solution advice

platforms like WhatsApp as ‘social media’) while 82.3% stated *No*, (Fig. 6b). Students were also asked if they frequently used social media. Most of them said ‘yes’, whereby 81.3% of the answers were placed between 6 and 10 on the answer-scale. A minority of 19% used social media rarely, (scale: 10 = ‘very frequent’—1 = ‘not at all’). 76.1% claimed to be frequent on Facebook, 66.4% on Instagram, and 44.2% on Twitter, whereby some students used all three, (Fig. 5). They were also asked how frequently they access their student-mailbox compared to how frequently they use social media: only 8.8% claimed to regularly check their e-mails; 43.4% claimed to check their e-mails 2–5 times per day, while the rest checked e-mails only once per day or rarely at all, (Fig. 6c). When the students were asked if they preferred class feedback and learning on social media, 86.7% of them agreed. Part of that majority preferred that our tool to be implemented on other social platforms such as Facebook or WhatsApp which would allow them to be continuously exposed to programming concepts, (Fig. 6d). Hence we conclude that automating course feedback on social media may be beneficial for continuous learning outside the classrooms.

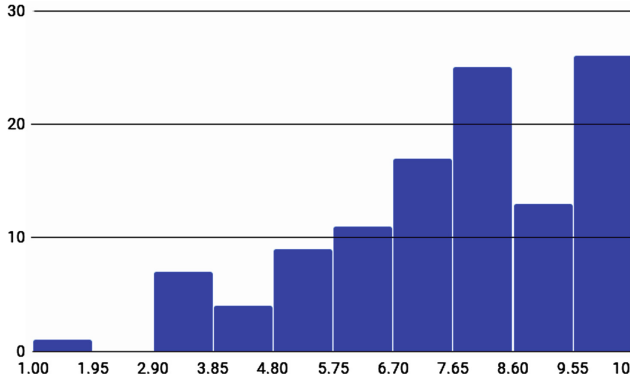


Fig. 5. Survey results: frequency of social media usage

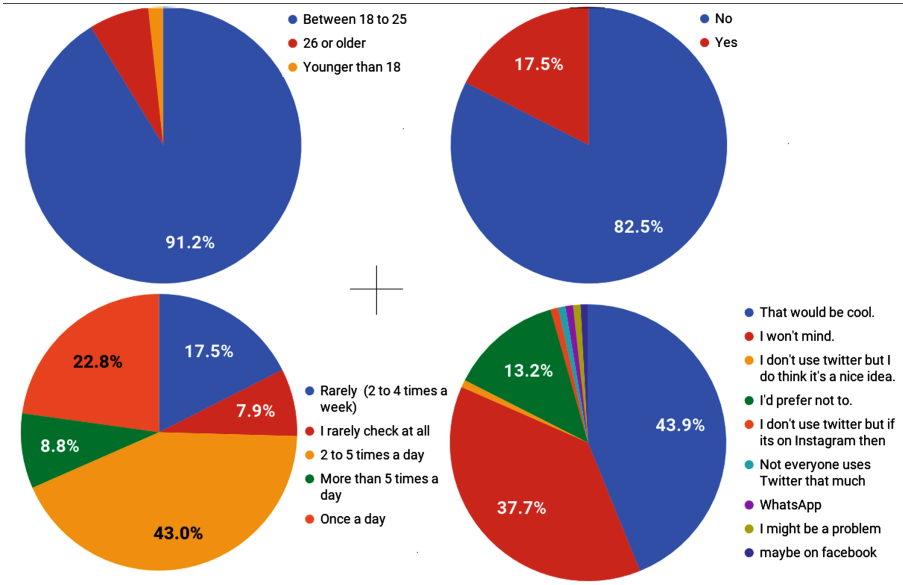


Fig. 6. Survey results: (a) Age group, (b) Do instructors use social media in support of teaching, (c) Frequency of e-mail access, (d) Preference for educational announcements on social media

6 Conclusion and Future Work

In this paper we have presented a CFG for the automatic generation of social media messages and tweets for novice programmers. We demonstrated a method for automating feedback in education on social media. Not limited to Twitter, our technique can be used also on other popular platforms (e.g. Facebook). Our

evaluation shows that a majority of students in our survey are frequent social media users and would prefer this form of feedback for educational purposes.

In the future we want to explore the synthesis of educational feedback on other social media platforms. Furthermore we will investigate how we can use similar techniques in other areas such as marketing or gaming.

Acknowledgment. Many thanks to *Nikita Patel* for having drawn the pictures which appear in this paper.

References

1. Abe, P., Jordan, N.A.: Integrating social media into the classroom curriculum. *About Campus* **18**(1), 16–20 (2013)
2. Ade-Ibijola, A.: Syntactic generation of practice novice programs in python. In: Kabanda, S., Suleman, H., Gruner, S. (eds.) *SACLA 2018. CCIS*, vol. 963, pp. 158–172. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-05813-5_11
3. Ade-Ibijola, A.: Synthesis of regular expression problems and solutions. Accepted for publ. *Int. J. Comput. Appl.* (forthcoming)
4. Ade-Ibijola, A.: Synthesis of social media profiles using a probabilistic context-free grammar. In: *PRASA-RobMech Proceedings of Pattern Recognition Association of South Africa and Robotics and Mechatronics*, pp. 104–109. IEEE (2017)
5. Aho, A.V., Lam, M.S., Sethi, R., Ullman, J.D.: *Compilers — Principles, Techniques and Tools*, 2nd edn. Addison Wesley, Boston (2007)
6. Bauer, D., Koller, A.: Sentence generation as planning with probabilistic LTAG. In: *TAG+10 Proceedings of the 10th International Workshop on Tree Adjoining Grammar and Related Frameworks*, pp. 127–134 (2010)
7. Bergin, S., Mooney, A., Ghent, J., Quille, K.: Using machine learning techniques to predict introductory programming performance. *Int. J. Comput. Sci. Softw. Eng.* **4**(12), 323–328 (2015)
8. Bicen, H., Cavus, N.: Twitter usage habits of undergraduate students. *Procedia Soc. Behav. Sci.* **46**, 335–339 (2012)
9. Buettner, R.: The utilization of Twitter in lectures. In: *Proceedings of the GI-Jahrestagung*, pp. 244–254 (2013)
10. Chawinga, W.D.: Taking social media to a university classroom: teaching and learning using Twitter and blogs. *Int. J. Educ. Technol. High. Educ.* **14**(1), 3 (2017)
11. Danlos, L., Maskharashvili, A., Pogodalla, S.: Interfacing sentential and discourse TAG-based grammars. In: *TAG+12 Proceedings of the 12th International Workshop on Tree Adjoining Grammars and Related Formalisms*, pp. 27–37 (2012)
12. Dijkstra, E.W.: How do we tell truths that might hurt? Manuscr. EWD 498. In: Dijkstra, E.W. (ed.) *Selected Writings on Computing: A personal Perspective*. MCS, pp. 129–131. Springer, New York (1982). https://doi.org/10.1007/978-1-4612-5695-3_22
13. Gachago, D., Ivala, E.: Social media for enhancing student engagement: the use of Facebook and blogs at a university of technology. *S. Afr. J. High. Educ.* **26**(1), 152–167 (2012)
14. Gil de Zúñiga, H., Jung, N., Valenzuela, S.: Social media use for news and individuals' social capital, civic engagement and political participation. *J. Comput. Mediat. Commun.* **17**(3), 319–336 (2012)

15. Gomes, A., Mendes, A.J.: Learning to program: difficulties and solutions. In: Proceedings of the International Conference on Engineering Education, vol. 7, p. 5 (2007)
16. Hepplestone, S., Holden, G., Irwin, B., Parkin, H.J., Thorpe, L.: Using technology to encourage student engagement with feedback: a literature review. *Res. Learn. Technol.* **19**(2), 117–127 (2011)
17. Hussain, I.: A study to evaluate the social media trends among university students. *Procedia Soc. Behav. Sci.* **64**, 639–645 (2012)
18. Kaplan, A.M., Haenlein, M.: Social media: back to the roots and back to the future. *J. Syst. Inf. Technol.* **14**(2), 101–104 (2012)
19. Kaplan, A.M., Haenlein, M.: Users of the world, unite! The challenges and opportunities of social media. *Bus. Horiz.* **53**(1), 59–68 (2010)
20. Kassens-Noor, E.: Twitter as a teaching practice to enhance active and informal learning in higher education: the case of sustainable tweets. *Act. Learn. High. Educ.* **13**(1), 9–21 (2012)
21. Kumagai, K., Kobayashi, I., Mochihashi, D., Asoh, H., Nakamura, T., Nagai, T.: Human-like natural language generation using monte carlo tree search. In: Proceedings of the INLG Workshop on Computational Creativity in Natural Language Generation, pp. 11–18 (2016)
22. Lahtinen, E., Ala-Mutka, K., Järvinen, H.M.: A study of the difficulties of novice programmers. *ACM SIGCSE Bull.* **37**(3), 14–18 (2005)
23. Malik, S.I.: Role of ADRI model in teaching and assessing novice programmers. Technical report, Deakin Univ. (2016)
24. Malik, S.I., Coldwell-Neilson, J.: A model for teaching and introductory programming course using ADRI. *Educ. Inf. Technol.* **22**(3), 1089–1120 (2017)
25. Mangold, W.G., Faulds, D.J.: Social media: the new hybrid element of the promotion mix. *Bus. Horiz.* **52**(4), 357–365 (2009)
26. Menkhoff, T., Chay, Y.W., Bengtsson, M.L., Woodard, C.J., Gan, B.: Incorporating microblogging tweeting in higher education: lessons learnt in a knowledge management course. *Comput. Hum. Behav.* **51**, 1295–1302 (2015)
27. Milne, I., Rowe, G.: Difficulties in learning and teaching programming: views of students and tutors. *Educ. Inf. Technol.* **7**(1), 55–66 (2002)
28. Murthy, D.: Twitter. Wiley, Hoboken (2018)
29. Obaido, G., Ade-Ibijola, A., Vadapalli, H.: Generating narrations of nested SQL queries using context-free grammars. In: Proceedings of the ICTAS Conference on Information Communications Technology and Society, pp. 1–6. IEEE (2019)
30. Paris, C.L., Swartout, W.R., Mann, W.C. (eds.): *Natural Language Generation in Artificial Intelligence and Computational Linguistics*. Springer, New York (1991). <https://doi.org/10.1007/978-1-4757-5945-7>
31. Reiter, E., Dale, R.: *Building Natural Language Generation Systems*. Cambridge University Press, Cambridge (2000)
32. Renumol, V., Jayaprakash, S., Janakiram, D.: Classification of Cognitive Difficulties of Students to Learn Computer Programming. Technical report, Indian Institutes of Technology (2009)
33. Robins, A., Rountree, J., Rountree, N.: Learning and teaching programming: a review and discussion. *Comput. Sci. Educ.* **13**(2), 137–172 (2003)
34. Roblyer, M.D., McDaniel, M., Webb, M., Herman, J., Witty, J.V.: Findings on Facebook in higher education: a comparison of college faculty and student uses and perceptions of social networking sites. *Internet High. Educ.* **13**(3), 134–140 (2010)

35. Ryan, J., Seither, E., Mateas, M., Wardrip-Fruin, N.: Expressionist: an authoring tool for in-game text generation. In: Nack, F., Gordon, A.S. (eds.) ICIDS 2016. LNCS, vol. 10045, pp. 221–233. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-48279-8_20
36. Small, T.A.: What the hashtag? A content analysis of canadian politics on Twitter. *Inf. Commun. Soc.* **14**(6), 872–895 (2011)
37. Sobaih, A.E.E., Moustafa, M.A., Ghandforoush, P., Khan, M.: To use or not to use? Social media in higher education in developing countries. *Comput. Hum. Behav.* **58**, 296–305 (2016)
38. Tan, P., Ting, C., Ling, S.: Learning difficulties in programming courses: undergraduates' perspective and perception. In: Proceedings of the International Conference on Computer Technology and Development, pp. 42–46 (2009)
39. Tang, Y., Hew, K.F.: Using Twitter for education: beneficial or simply a waste of time? *Comput. Educ.* **106**, 97–118 (2017)
40. Twitter: Post, Retrieve and Engage with Tweets. <https://developer.twitter.com/en/docs/tweets/post-and-engage/overview>



Decoding Source Code Comprehension: Bottlenecks Experienced by Senior Computer Science Students

Pakiso J. Khomokhoana[✉] and Liezel Nel[✉]

Department of Computer Science and Informatics, University of the Free State,
Bloemfontein, South Africa
nell@ufs.ac.za

Abstract. Source code comprehension (SCC) continues to be a challenge to undergraduate CS students. Understanding the mental processes that students follow while comprehending source code can be crucial in helping students to overcome related challenges. The ‘Decoding the Disciplines’ (DtDs) paradigm that is gaining popularity world-wide provides a process to help students to master the mental actions they need to be successful in a specific discipline. In focusing on the first DtDs step of identifying mental obstacles (“bottlenecks”), this paper describes a study aimed at uncovering the major SCC bottlenecks that senior CS students experienced. We followed an integrated methodological approach where data were collected by asking questions, observations, and artefact analysis. Thematic analysis of the collected data revealed a series of SCC difficulties specifically related to arrays, programming logic, and control structures. The identified difficulties, including findings from the literature as well as our own teaching experiences, were used to compile a usable list of SCC bottlenecks. By focusing on senior students (instead of first-year students), the identified SCC bottlenecks point to learning difficulties that need to be addressed in introductory CS courses.

Keywords: Computer programming · Source code comprehension · Students’ learning bottlenecks · Decoding the Disciplines

1 Introduction

Despite the continuous efforts of committed instructors to share the intricacies of their academic disciplines and their students’ desperation to succeed, many students still struggle to master course material [32]. The specific points where students’ learning gets interrupted can be referred to as *bottlenecks* [11, 28]. A bottleneck typically occurs when students are unsure about how to approach a problem and consequently follow inappropriate strategies [32]. In an attempt to assist instructors in addressing students’ learning bottlenecks, Middendorf and Pace devised the *Decoding the Disciplines* (DtDs) paradigm [28]. One of the underlying principles of this paradigm is that each discipline has unique

ways of thinking [28]. Students who fail to master the required ‘ways of thinking’ are unlikely to succeed in their higher-level studies. In the DtDs paradigm, instructors are therefore encouraged to identify discipline-specific learning bottlenecks that could prevent students from mastering the basic disciplinary ways of thinking. Subsequently, specific strategies to address the bottlenecks are identified, implemented and evaluated [32]. Despite the recent uptake in decoding research in other disciplines [40, 43], limited information about DtDs research in Computer Science (CS) is available.

However, during the past three decades numerous investigations have been launched to gain better understanding of the various difficulties that computer programming students experience [3, 4]. One such difficulty—which has been studied extensively—relates to the way in which students (also referred to as ‘novice programmers’) interpret pieces of source code [9, 23]. This activity—commonly referred to as *source code comprehension* (SCC)—is regarded a vital skill that novice programmers have to master [39]. Most of the previous SCC studies, however, focused on the evaluation of difficulties that students enrolled for introductory programming courses experience [26, 38]. Pace points out that a student’s inability to master certain basic concepts may not necessarily lead to his/her failure of an introductory course [32]. However, it is likely that such a student’s confusions will continue to accumulate, thus causing diminishing performance of basic tasks. As such, it is possible for students to progress to advanced courses while they are still experiencing bottlenecks related to basic concepts. Their failure to grasp these basic concepts can have a negative impact on their ability to complete their degrees. This paper therefore attempts to answer the following two *questions*:

1. What are the major SCC difficulties experienced by senior CS students?
2. How can knowledge of these difficulties be used to identify SCC bottlenecks that should ideally be addressed in introductory programming courses?

In the remainder of this paper, a review of relevant background literature is presented in Sect. 2. This is followed by a discussion of the research design and method in Sect. 3, and a presentation and interpretation of the results in Sect. 4. The identified SCC bottlenecks are presented in Sect. 5, and conclusions and recommendations for future research in Sect. 6.

2 Related Work

The first step of the seven-step DtDs framework [28] is to identify students’ learning bottlenecks. The identification of discipline-specific bottlenecks allows instructors to identify specific areas in a module where they need to intervene more strongly in order to facilitate better learning [29, 32]. In identifying a learning bottleneck, the instructor must ensure that the bottleneck is ‘useful’. A bottleneck is ‘useful’ if it affects the learning of *many* students, interferes with the major learning in a module, is relatively focused, and does not involve a large number of disparate operations. It must also be defined clearly without jargon [32]. In the DtDs paradigm [28], instructors can take various ways to identify bottlenecks.

2.1 Bottleneck Identification Approaches

In one of the popular approaches [29], instructors themselves identify bottlenecks based on specific student problems they discover during their teaching of a specific module [34]. Instructors can also identify bottlenecks by focusing on a single assignment. In History, for example, Pace identified a specific difficulty while grading a writing assignment [32], whereas Shopkow was alerted to a specific difficulty as a result of questions voiced by her students regarding the specifications of an assignment [40].

In most of the limited number of decoding studies in the CS discipline to date, researchers have also identified specific bottlenecks based on personal teaching experiences. For a Database Design and Data Retrieval module, the authors of [19] identified creating Entity Relationship diagrams, reasoning in MySQL and dualism as the main student learning bottlenecks. Menzel used her experience in teaching an introductory CS module to identify recursion — a threshold concept in CS [38]— as her students’ main bottleneck [27]. For a follow-up module, German focused his decoding study on the challenges of program debugging [14].

Bottleneck identification for a specific module can also be facilitated by an outsider (e.g. a pedagogical advisor). In [43], module-specific bottlenecks were identified by asking seven participants representing five disciplines (Engineering, Chemistry, History, Social Sciences and Electronics) to each write a 10-line description of two or three bottlenecks they could think of for the modules they were teaching. In an attempt to identify the top bottlenecks experienced by Accounting students in their Taxation modules, Timmermans and Barnett first asked instructors to identify potential bottlenecks [42]. Their eventual selection of the top bottlenecks was based on the responses of 4th-year Taxation students who were asked to rate the 40 potential bottlenecks w.r.t. level of understanding and importance.

When the goal is to identify common bottlenecks in a specific discipline, the collective experiences of a group of instructors can also be a valuable source. In this regard, various researchers from the History discipline have used individual interviews with instructors to identify common discipline-specific bottlenecks [11, 41]. Wilkinson chose a peer dialogue strategy where Law instructors collectively established that the reading of case law was their students’ major learning bottleneck [46]. For bottleneck identification in Political Science, Rouse (et al.) based their selection of literature reviews as the major bottleneck on the experiences of both instructors and students (from different year levels) as well as on the findings of other studies [37]. Thus an instructor’s insight often is the main source for bottleneck identification. However, the role of students in bottleneck identification should not be ignored. Further justification for the seriousness of specific bottlenecks can also be found by linking bottlenecks to discipline-specific learning difficulties identified in other non-decoding studies (such as [31]).

2.2 SCC Difficulties

As mentioned above, numerous previous studies have attempted to uncover the specific difficulties experienced by novice programmers in comprehending source

code. Although none of these studies were specifically conducted in the DtDs framework, Middendorf and Shopkow suggest that relevant literature can also be used to identify bottlenecks [29].

In an investigation of the programming competency of students enrolled for CS1 and CS2 courses, McCracken (et al.) stated that many students still do not know how to program at the end of their introductory programming courses [26]. This problem was further explored in the BRACElet project which confirmed students' lack of programming skills [45]. In an attempt to expand understanding of the difficulties experienced by students, [26] refers to the potential role that in-depth analysis of narrative data collected from students can play.

The ITiCSE 2004 working group study [23] was conducted as a follow-up on the McCracken (et al.) study. A set of 12 multiple-choice questions (MCQs) was used to test students' ability on two tasks: firstly, to predict the output of executing the given fragments of source code; secondly, to select a piece of source code (from a small set of options) that would correctly complete a given near-complete code snippet. Although many students were found to be lacking the skills required to do both tasks, the latter was found to be the most challenging. The final ITiCSE 2004 report states that students were unable to "*reliably work their way through the long chain of reasoning required to hand-execute code, and/or... to reason reliably at a more abstract level to select the missing line of code*" [23] (p. 132).

All 12 questions used in [23] focused strongly on the concept of *arrays*. In a study aimed at improving students' learning experiences, Hyland and Clynych found arrays to be the most challenging topic for first- and second-year students [18]. In an attempt to record all the difficulties that students experience during practical computer programming sessions, Garner (et al.) found arrays to be featuring among the top three difficulties [13]. Other studies, too, have identified arrays as a challenging concept for novice programmers [2, 24].

All ITiCSE 2004 questions [23] included basic *control structures* such as conditionals (e.g. `if`, `if-else`), loops (e.g. `while`, `for`), or a combination thereof. According to [30], many novice programmers struggle to comprehend basic control structures. Various studies have described the specific difficulties that students experienced while interpreting looping (repetition) structures [6, 16, 18, 24]. Garner (et al.) mention that most of the difficulties associated with loops originate in students' incorrect comprehension of either the header or body of the looping structure [13].

Although logic generally is regarded as a mathematical field, it has grown more relevant to CS especially w.r.t. its applications [17]. Programming logic involves executing statements contained in a given piece of code one after another in the order in which they are written. Though still logical and correct, there are some programming control structures that may violate this execution order [10]. It is therefore not surprising that students struggle with logical reasoning in solving computer programming related problems [6]. The logical flow of the source code statements is closely related to the control flow of such statements [13]. This implies that for a programmer to fully comprehend a computer program, he/she must skilfully combine the programming logic with the control flow

of the program. Students are more likely to logically work (or trace) through a piece of source code if they have adequate knowledge of the semantics of the programming language and can keep track of changes made to variable values [23]. Therefore novices especially struggle to follow a program’s execution [4,36] and control flow [13].

As the proponents of the DtDs paradigm argue that bottlenecks directly relate to difficulties hindering *many* students’ learning [28], the previously identified difficulties can serve as a baseline for the identification of common and useful SCC bottlenecks. The exact nature of some of these difficulties, however, remains unclear: Where exactly are students getting stuck? Why are they getting stuck? What are they doing wrong? Which strategies do they resort to when they get stuck? Better knowledge about the nature of these difficulties can thus be valuable in determining teaching and learning gaps related to SCC.

3 Research Methods

3.1 Design

Within the scope of a DtDs-based research design, we followed an approach based on Plowright’s Frameworks for an Integrated Methodology (*FraIM*) [35]. Thereby, our focus was on collecting narrative and/or numeric data by means of observations, asking questions, and/or artefact analysis. The study population consisted of final-year undergraduate CS students (referred to as ‘senior students’ in this paper) from a South African university. The empirical part of our study comprised two phases. The aim of Phase 1 was to identify specific senior CS students having trouble in comprehending short pieces of source code. In Phase 2 we wanted to detect specific points or ‘places’ [28] where these students were experiencing SCC difficulties, with the goal of identifying common and useful SCC bottlenecks.

3.2 Phase 1

Participants, Data Collection and Analysis. The sample for Phase 1 consisted of 40 students registered for a 3rd-year Internet Programming module. The selection of this sample was both ‘purposeful’ and ‘convenient’ [33]. The sample was purposeful because the students had already completed four earlier programming modules. However, they could still be regarded as ‘novice’ programmers since they did not yet have any professional programming experience. The sample was also convenient since we had easy access to the participants because the lecturer responsible for this module agreed to make available one of her scheduled class sessions for our research activity.

For the research activity of Phase 1, participants were given a test consisting of the 12 questions of [23]. For each of these questions, participants had to work through a short fragment of source code, and then either predict the execution output of the code fragment or select (from a small set of options) the relevant piece of code needed to complete the given fragment. The questions of [23]

were chosen for two reasons: Firstly, all of them contained source code fragments that students had to comprehend before they could answer the related question. Secondly, these questions had already been tested with a large population of students from several universities in the USA and other countries.¹ Since the original questions were formulated in Java, we had to convert their code fragments to C# because this is the programming language familiar to our chosen population.

The participants' answer sheets (regarded as 'artefacts') were our primary data source for Phase 1. After 'grading' the artefacts, the performance data for each participant were captured into a spreadsheet, and descriptive statistics were used to rank the questions in order of difficulty (based on the number of participants who incorrectly answered such question). The three apparently most difficult questions (Q3, Q6, Q8) were chosen for further use in Phase 2.

3.3 Phase 2

Data Collection. Based on the student performance data collected in Phase 1, 15 students were invited to take part in Phase 2. These were the students who provided wrong answers to all three of the questions identified in Phase 1. Ten of the 15 invited students agreed to partake in Phase 2. The research activity in Phase 2 consisted of individual sessions during which each participant had to verbally expose his/her thinking process(es) in a form of 'thinking aloud' [7] while answering anew the same three above-mentioned SCC questions. This data collection strategy can be regarded as a means of 'asking questions'.

Time slots of 45 min were scheduled for each individual session. However, the participants were informed that they could take as much time as they needed to complete the tasks. Since none of the participants had prior experience with the required think-aloud technique, this technique was first demonstrated to each participant on an unrelated SCC question. Thereby, one of us played the role of the 'interviewer' by asking probing questions when required (i.e. no progress or silence). Where necessary we also recorded some observations, as an additional means of data collection, after the permission for audio-recording was obtained by the corresponding participant.

Data Analysis. To transcribe and analyse the audio recordings from the individual think-aloud sessions, we followed the approach of [8]. Upon data transcription we 'cleansed' the data by searching for faults and by repairing them accordingly [47]. Since the participants had to verbalise their thoughts as part of the think-aloud process, the transcripts contained numerous illogical and repeated statements. We therefore decided to use 'fuzzy validation' (instead of strict validation which requires the complete removal of invalid or undesired responses) [47]. In fuzzy validation we are allowed to correct some data if there is a reasonably 'close match' to a known right answer. Thereafter we familiarised ourselves with the data [25] by listening and re-listening to the audio records numerous

¹ *Benchmark* for international comparability.

times as well as by intensively and repeatedly reading the transcripts. This helped us to devise a coding plan in which the analysis would be guided by the data related to our research questions. At this stage, the 10 validated transcripts were imported into the Nvivo 12 Professional tool. Thereafter, codes were developed (by creating several nodes) for each SCC difficulty identified in the data.

For coding, Klenke recommends the use of ‘units of analysis’ [22]. These can be words, sentences or paragraphs. Accordingly, we coded the data by highlighting and/or underlining text (from which the SCC difficulties could be extracted) within the domain of the stated units of analysis. Then we ‘populated’ the created codes by associating the corresponding texts with them. During this process of refinement, the names of the codes were continuously revised until *relevant themes* began to emerge. For each emerging theme, its Nvivo-generated frequency of occurrence was taken into account.

4 Results and Interpretation

Given the large amount of data collected during Phase 2, the results discussion only focuses on the participants’ comprehension of Question 3 (Fig. 1, with line numbers 1–12 inserted in aid of this discussion). This question was selected since its related ‘think-aloud’ data revealed most clearly the numerous difficulties that can be directly associated with SCC. Q3 also tested students’ comprehension of arrays and basic control structures—i.e. the concepts previously identified as challenging for novice programmers (see above). In the following discussion, the eight most common SCC difficulties identified are grouped into *three categories*: arrays, programming logic, and control structures.

4.1 Array-Related Difficulties

Analysis of the Q3 think-aloud data revealed the following *four* major array-related difficulties experienced by the participants.

Array Index. An array index refers to a non-negative integer number that identifies the position of an element stored in an array. Four participants had difficulties to interpret simple array indices, with a total of nine occurrences identified. Participant 1 (P1) had the most difficulties in this regard, with three occurrences identified. In her interpretation of `b[i]`, she regarded `i` as a value contained *in* array `b` instead of a position *of* an element. Participant P8 confused the square brackets indicating the array index with a multiplication operator when he interpreted `b[i]` as `b` multiplied by `i`: “*int i is equal to 0 [Line 8], and then for **this times that**, it is equal to true [Line 10] then increment the **counter** [Line 11], **that times that** is equal to true ... it is a difficult one but then ... **that times that** is true and **that times that** is true”.* Thus, both participants were challenged by the *notation* [20] of the array index.

Array Length. The length of an array refers to the maximum number of values that can be stored in it. Three participants struggled to determine the length of

Question 3
Consider the following source code fragment:

```

1.  int[] x = {1, 2, 3, 3, 3};
2.  bool[] b = new bool[x.Length];

3.  for (int i = 0; i < b.Length; ++i)
4.      b[i] = false;

5.  for (int i = 0; i < x.Length; ++i)
6.      b[x[i]] = true;

7.  int count = 0;

8.  for (int i = 0; i < b.Length; ++i)
9.  {
10.     if (b[i] == true)
11.         ++count;
12. }

```

After this source code is executed, **count** contains:

- a) 1
- b) 2
- c) 3
- d) 4
- e) 5

Fig. 1. Question 3 according to [23]

the arrays in Q3. P1 had no idea how to determine the length of the Boolean array **b** and remarked: “*I do not know what is the length of array b*”. Also P6 was unable to correctly determine the length of the array. He interpreted the Boolean array **b** to have the length of 4 (instead of the correct length 5): “*So now is 0 less than 4 because our b value is 4*” while looking at the condition of the for loop in [Line 3].

Boolean Array. A Boolean array is an array of which the elements can only contain the values *true* or *false*. Five occurrences of Boolean array difficulties were identified, whereby P7 was most challenged (with three identified occurrences). Overall, the identified difficulties ranged from the declaration of the Boolean array to basic understanding regarding the effects of operations performed on such arrays.

P7 got stuck at the Boolean array declaration in [Line 2] and skipped the question: “*Do I understand what I am doing? ...it is a Boolean array, array is a Boolean, what does it mean? ... (pause) ... I am not sure about this one yet, let me...*” (turning the page to see the next question). When P7 returned to this question later, his confusion regarding Boolean arrays became even more apparent as he regarded the index value of 1 as the Boolean equivalent of *true*: “*Once it gets to the if statement, i is now equal to 1 and 1 is equal to true*” [Line 10].

P9 was under the impression that since `b` was a Boolean array it could only consist of two elements (instead of two possible values per element): *“In position 0, I have 1, which means now at `b[1]` I have `true`. In my `bool` array I have stored two values”* [Line 10].

In their comprehension of [Line 10], both P7 and P9 disregarded the actual code *syntax*. Instead, they fell back to some *semantic* according to which a 0 represents `false` and a 1 represents `true`. Both participants believed the numeric index positions 0 and 1 to represent their Boolean truth value equivalents.

Decomposition—whereby a complex piece of code is ‘split’ into its constituent components to simplify its interpretation [21]—is a task with which many novice programmers struggle [15]. In their comprehension of Q3, seven participants found it particularly difficult to decompose the compound index in the expression `b[x[i]]` of [Line 6]. Altogether 29 occurrences of this difficulty were identified.

P10 misinterpreted [Line 6] as ‘resetting’ *all* values in `b` to `true`, whereas de-facto only the selected values in array `b` are set to `true`: *“`b[x[i]]` set to true [Line 6]... yes, no, I am very confused ...(longer pause)... `b[i]` ...then the second `for` loop [Line 5] sets everything from the integer array to `true`, so, if I am correct, then it resets everything from the first `for` loop [Line 3] back to `true`”*.

P6 became so confused with the meaning of the compound index expression that he could not even grasp how the code in [Line 6] was related to the `for` loop in [Line 5]: *“Now I am worried about this `for` loop, the second `for` loop [Line 5], it seems like it has nothing to do with the rest of the statements that come after it... so this second `for` loop is the one that is freaking me out”*. Although P6 had no difficulty to understand any of the other `for` loops in Q3, it seems that his inability to decompose the compound index expression caused so much confusion that he suddenly could not comprehend the basic execution of the `for` loop in [Line 5].

4.2 Programming Logic Difficulties

The following *three* programming logic difficulties were identified from our Q3-related think-aloud transcripts.

Ripple Effect. This effect occurs when the misinterpretation of one statement has a direct impact on the interpretation of statements that follow. This difficulty, which was observed with three participants, typically arises when programmers misinterpret programming logic [20]. Due to P1’s failure in interpreting array indices (see above), her interpretation of the statements contained in the third `for` loop completely ignored any changes made to the elements of `b` in the first two `for` loops [Lines 3–6]. She remarked: *“If `b[i]` is `true` [Line 10], I increment `count` [Line 11]. So if I increment `count` every time until it is over 5, then I will have 5”*. Thus she wrongly chose ‘5’ (option `e`) as her answer to Q3.

The difficulties that P6 had in interpreting the second `for` loop [Lines 5–6] (see above: Decomposition) caused him to disregard that loop entirely while interpreting the third `for` loop: “*When looking at this third for loop [Line 8], it is the same as the first one [Line 3] that says the bool array is always equal to false. Now in the third one they are saying if the element at position i in the Boolean array is equal to true [Line 10], then increment count [Line 11]. But according to this [Line 4], the b value is always false*”.

The utterances by both P1 and P6 indicated that they were not thinking sequentially [5], and therefore failed to follow the algorithmic logic of the source code in question [1]. P9 showed similar behaviour after she realised that she could not interpret any of the `for` loops and the containing statements. In response, she reverted her attention to those statements that she could comprehend and only considered those to arrive at `count=1` as her answer to Q3. Her non-sequential (non-algorithmic) reasoning appears in the following excerpt: “*My first index: I have a false [Line 4], and then my second: I have a true [Line 6], and then int count is equal to 0 [Line 7] ...it will only increment when I get to this point [Line 11], whereby count needs to be 1*” (option a).

The most concerning aspect of the thinking patterns portrayed by these participants is the ‘mental block’ caused by the statements they could not fully comprehend, and their consequent anxious behaviour (observed by the interviewer). These participants tried to ‘escape’ the block by entirely ignoring the troublesome statements *as if* those were no longer part of the given code.²

Guessing. A common critique of MCQs is that they are solvable through guessing. This is also true of our 12 MCQs (Part 1) for which guessing behavior was previously observed [12, 23]. However our format of think-aloud sessions (Phase 2) discouraged guessing, because the participants were repeatedly prompted to explain their reasoning in as much detail as possible. Nonetheless, one participant (P8) attempted guessing when saying “*I just have to go with option a*” after having traced only a small fraction of the given code. At that stage he was unable to show how he arrived at the chosen answer and had to be prompted by the interviewer to re-explain his reasoning.

Mathematical Expressions. When a line of code contains a mathematical expression, the misinterpretation of an operator can interfere with the comprehension of program logic; for comparison see [31]. One example of such a mistake was observed when P7 failed to grasp the execution of the third `for` loop [Line 8] when the value of `i` increased to 5: “*Yes, i becomes 5... once it runs throughout the loop and becomes 5 then... b[i] is going to be true... then the count also increments*”. He thus treated the `<` as if it was a `≤` operator, which is typically regarded as a ‘logical error’ in the comprehension of source code.

² This observation points to the *psychology* of programming which is a field of research since the late 1960s/early 1970s [44].

4.3 Programming Control Structure Difficulty

The code in Q3 only contained one type of control structure in the form of three `for` repetition structures. As mentioned above (Ripple Effect), the lack of understanding that P6 and P9 portrayed regarding the overall functioning of a `for` loop caused them to eventually ignore the lines of code related to these structures. Another `for` loop misconception was observed when P7 repeatedly executed the loop counter increment statement (`++i`) at the beginning of each loop, thereby setting the initial value of `i` to 1 for each of the three loops. Since repetition structures are one of the concepts that novices find challenging [16], it is not surprising that some participants experienced difficulties in this regard. However, one area of concern is the level of difficulty that these *senior* students experienced in comprehending *basic* `for` repetition structures.

5 Identification of Six SCC Bottlenecks

The results of Phase 2 showed that the participants in this study experienced eight major SCC difficulties related to the concept of arrays, programming logic, and programming control. Following the bottleneck identification guidelines of [29,32], we used our collective experience of more than 25 years of teaching introductory and advanced programming courses, combined with the insights gained from this study as well as relevant literature, to formulate the following six ‘usable’ SCC bottlenecks.

5.1 Bottleneck 1

Students cannot keep track of variable values while tracing through a piece of code. The above-mentioned think-aloud excerpts contain numerous examples in which students lost track of the changes made to variable values. The students tried to remember the changes to the variable values (instead of writing notes on a provided piece of paper), which put unnecessary strain on their working memories. Their incorrect answers were thus a direct consequence of failing memory or guessing. Lister (et al.) pointed out that, when students document changes to variable values, they are much more likely to produce the correct answer [23]. Most students in our study did not follow a reliable strategy to keep track of such value changes.

5.2 Bottleneck 2

Students cannot comprehend statements containing arrays and basic operations on array elements. The bulk of the identified difficulties can be related to the students’ wrong understanding of array concepts: for comparison see [2,13,18,24]. Our students particularly struggled to interpret array indices—especially in combination with other concepts. While one student confused the square [index] brackets with a multiplication operator, others were not able to

determine an array's length. Although most students had little trouble to comprehend the array of numeric values, many of them were lost when having to deal with the Boolean array type.

5.3 Bottleneck 3

Students cannot comprehend the execution of basic for repetition structures. Most of the difficulties observed with the `for` loops are due to our students' incorrect comprehension of either the header or the body of the looping structure (as in [13]). While some students failed to recognise when and how a loop terminates (see also [16]), one instance was observed where the loop counter increment statement was executed at the wrong time. Although most of the difficulties observed in comprehension of the body of the looping structure are more specifically related to arrays, referencing the incorrect value of the loop counter variable also caused problems for some students. Most worrying were the two students who entirely gave up interpreting the `for` loops and ignored either the entire structure or the loop header for the remainder of their Q3 interpretation.

5.4 Bottleneck 4

Students do not have adequate strategies to interpret lines of code that look unfamiliar. This bottleneck was observed in cases where students were unable to read, interpret and understand (execute) a specific code statement. Of particular interest here are cases where two or more separate concepts—which a student could comprehend earlier—were combined to form one 'complex' concept. The students were unable to decompose the more complex piece of code into smaller parts in order to simplify the interpretation thereof, (see also [21]). Their most common response to this challenge was to ignore those complex statements or lines of code. Although decomposition is a task that many novice programmers struggle with [15], students may never learn how to deal with complex concepts if they are not taught explicit strategies to resort to in such situations.

5.5 Bottleneck 5

Students view a piece of source code as consisting of separate lines of code, thereby ignoring the significance of each individual line. We typically teach our students that, in order to fully comprehend what a program does, they first need to understand the meaning of each distinct line of code of that program. However, it seems that in following our 'guidelines' some students not only lose sight of how the parts fit together but also of the overall significance of each individual line of code or statement. This behaviour was evident with those students who completely ignored sections of code they could not comprehend, with a complete disregard for the impact this would have on their ability to

determine the correct answer to Q3. For comparison see the ‘*ignore significance*’ bottleneck in [41] about History students’ disregard of how individual facts relate to the story they are trying to tell.

5.6 Bottleneck 6

Students cannot reliably think their way through a long chain of reasoning required to comprehend a piece of source code. This bottleneck can be regarded as ‘overarching’ since it refers to one of the most common and significant SCC difficulties identified in [23]. In our study it is directly related to our ‘*ripple effect*’ that refers to mistakes made when students are unable to think sequentially [5] or fail to follow the source code logic [1]. In our study, we noticed the significant negative impact that inadequate knowledge of semantics and inability to keep track of variable values can have on a student’s comprehension of a piece of code. These are all examples of actions that can cause a ‘mental block’ in students’ reasoning ability, which they are unlikely to overcome if they do not have the necessary knowledge and abilities to deal with such difficulties.

Although we present these as six separate bottlenecks, they should be seen as “*interconnected with each other*” [41], since they are all indicators of mental challenges experienced by novice programmers while comprehending source code.

6 Conclusions and Future Work

SCC continues to be a challenge to undergraduate CS students. Understanding the mental processes that students follow while comprehending source code can be crucial in helping students to overcome related challenges. By focusing on Step 1 of the seven-step DtDs framework, this study aimed to expose the major SCC bottlenecks experienced by senior CS students. Thematic analysis of data collected by means of asking questions, observations, and artefact analysis revealed several SCC difficulties specifically related to arrays, programming logic and control structures. The detected difficulties, combined with findings from the literature and our personal experiences, were used to formulate six bottlenecks that are indicative of the typical mental challenges experienced by novice programmers during the comprehension of source code. By focusing on senior students we were able to identify major bottlenecks that point to students’ learning difficulties that are currently not adequately addressed in introductory CS courses, and therefore continue to influence the mental processes of final-year undergraduate students.

With this paper we also wanted to raise awareness among instructors regarding the role that a systematic ‘decoding’ approach can play in exposing the mental processes and bottlenecks unique to the CS discipline. To address the remaining six steps of the DtDs framework [28], future research is needed, firstly, to uncover the mental activities of *expert* programmers to overcome the identified SCC bottlenecks. This knowledge could then be used to devise teaching and

learning strategies that model the mental strategies of the experts. After creating opportunities for students to practice these skills and to receive feedback on their efforts, instructors can assess students' efforts to determine whether they have benefited from the implemented strategies or not. The ultimate goal of this suggested research is to help students to master the mental actions they need to be successful in the CS discipline.

References

1. Alston, P., Walsh, D., Westhead, G.: Uncovering 'threshold concepts' in web development: an instructor perspective. *ACM Trans. Comput. Educ.* **15**(1), 1–18 (2015)
2. Anyango, J.T., Suleman, H.: Teaching programming in Kenya and South Africa: what is difficult and is it universal? In: *Proceedings of the 18th Koli Calling International Conference on Computing Education Research*. ACM (2018)
3. Bosse, Y., Gerosa, M.A.: Difficulties of programming learning from the point of view of students and instructors. *EEE Lat. Am. Trans.* **15**(11), 2191–2199 (2017)
4. du Boulay, B.: Some difficulties of learning to program. *J. Educ. Comput. Res.* **2**(1), 57–73 (1986)
5. Boustedt, J., et al.: Threshold concepts in computer science: do they exist and are they useful? In: *Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education*, pp. 504–508. ACM (2007)
6. Butler, M., Morgan, M.: Learning challenges faced by novice programming students studying high level and low feedback concepts. In: *Proceedings Ascilite Singapore*, pp. 99–107 (2007)
7. Charters, E.: The use of think-aloud methods in qualitative research: an introduction to think-aloud methods. *Brock Educ.* **12**(2), 68–82 (2003)
8. Creswell, J.W., Creswell, J.D.: *Research Design: Qualitative, Quantitative, and Mixed Methods Approaches*. SAGE, Thousand Oaks (2017)
9. Cunningham, K., Blanchard, S., Ericson, B., Guzdial, M.: Using tracing and sketching to solve programming problems: replicating and extending an analysis of what students draw. In: *Proceedings of the International Conference on Computing Education Research*, pp. 164–172. ACM (2017)
10. Deitel, P.J., Deitel, H., Deitel, A.: *Visual Basic 2012 — How to Program*. Pearson Education, London (2013)
11. Diaz, A., Middendorf, J., Pace, D., Shopkow, L.: The history learning project: a department 'decodes' its students. *J. Am. Hist.* **94**(4), 1211–1224 (2008)
12. Fitzgerald, S., Simon, B., Thomas, L.: Strategies that students use to trace code: an analysis based in grounded theory. In: *Proceedings of the 1st International Workshop on Computing Education Research*, pp. 69–80. ACM (2004)
13. Garner, S., Haden, P., Robins, A.: My program is correct but it doesn't run: a preliminary investigation of novice programmers' problems. In: *Australasian Computing Education Conference*, pp. 173–180. Australian Computer Society Inc., Newcastle (2005)
14. German, A., Menzel, S., Middendorf, J., Duncan, F.J.: How to decode student bottlenecks to learning in computer science. In: *Proceedings of the 45th Technical Symposium on Computer Science Education*, p. 733. ACM (2014)
15. Goldman, K., et al.: Identifying important and difficult concepts in introductory computing courses using a delphi process. In: *Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education*, pp. 256–260. ACM (2008)





16. Grover, S., Basu, S.: Measuring student learning in introductory block-based programming: examining misconceptions of loops, variables, and boolean logic. In: Proceedings of the SIGCSE Technical Symposium on Computer Science Education, pp. 267–272. ACM (2017)
17. Gurevich, Y.: Logic and the challenge of computer science. In: Börger, E. (ed.) Current Trends in Theoretical Computer Science, pp. 1–57. Computer Science Press (1988)
18. Hyland, E., Clynych, G.: Initial experiences gained and initiatives employed in the teaching of Java programming in the Institute of Technology Tallaght. In: Joint Proceedings of the Inaugural Conference on the Principles and Practice of Programming, and 2nd Workshop on Intermediate Representation engineering for Virtual Machines, pp. 101–106. ACM (2002)
19. IUBCTL: Team-Based Learning For Practice and Motivation (2016). <https://www.youtube.com/watch?v=1obB-n6JZ8k>
20. Kallia, M., Sentance, S.: Computing teachers’ perspectives on threshold concepts: functions and procedural abstraction. In: Proceedings of the WIPSCS 12th Workshop on Primary and Secondary Computing Education, pp. 15–24 (2017)
21. Keen, A., Mammen, K.: Program decomposition and complexity in CS1. In: Proceedings of the 46th Technical Symposium on Computer Science Education, pp. 48–53. ACM (2015)
22. Klenke, K.: Qualitative Research in the Study of Leadership, 2nd edn. Emerald Publishing, Bingley (2016)
23. Lister, R., et al.: A multi-national study of reading and tracing skills in novice programmers. ACM SIGCSE Bull. **36**(4), 119–150 (2004)
24. Malik, S.I., Coldwell-Neilson, J.: A model for teaching an introductory programming course using ADRI. Educ. Inf. Technol. **22**(3), 1089–1120 (2017)
25. Marshall, C., Rossman, G.B.: Designing Qualitative Research, 6th edn. SAGE, Thousand Oaks (2016)
26. McCracken, M., et al.: A multi-national, multi-institutional study of assessment of programming skills of first-year CS students. In: Working Group Reports from ITiCSE on Innovation and Technology in Computer Science Education, pp. 125–180. ACM (2001)
27. Menzel, S.: ISSOTL 2015: Recursion as a Bottleneck Concept (2017). <https://www.youtube.com/watch?v=iNvQlm9phEI>
28. Middendorf, J., Pace, D.: Decoding the disciplines: a model for helping students learn disciplinary ways of thinking. New Dir. Teach. Learn. **98**, 1–12 (2004)
29. Middendorf, J., Shopkow, L.: Overcoming Student Learning Bottlenecks: Decode Your Disciplinary Critical Thinking. Stylus Publishing/LLC (2018)
30. Milne, I., Rowe, G.: Difficulties in learning and teaching programming: views of students and tutors. Educ. Inf. Technol. **7**(1), 55–66 (2002)
31. Mutanu, L., Machoka, P.: Enhancing computer students’ academic performance through explanatory modeling. In: Tait, B., et al. (eds.) SACLA 2019. CCIS, vol. 1136, pp. 227–243 (2020)
32. Pace, D.: The Decoding the Disciplines Paradigm: Seven Steps to Increased Student Learning. Indiana University Press (2017)
33. Patton, M.Q.: Qualitative Research & Evaluation Methods: Integrating Theory and Practice, 4th edn. SAGE, Thousand Oaks (2015)
34. Pinnow, E.: Decoding the disciplines: an approach to scientific thinking. Psychol. Learn. Teach. **15**(1), 94–101 (2016)
35. Plowright, D.: Using Mixed Methods: Frameworks for an Integrated Methodology. SAGE, Thousand Oaks (2011)

36. Qian, Y., Lehman, J.: Students' misconceptions and other difficulties in introductory programming: a literature review. *ACM Trans. Comput. Educ.* **18**(1), 1–24 (2017)
37. Rouse, M., Phillips, J., Mehaffey, R., McGowan, S., Felten, P.: Decoding and disclosure in students-as-partners research: a case study of the political science literature review. *Int. J. Stud. Partn.* **1**(1), 1–14 (2017)
38. Sanders, K., McCartney, R.: Threshold concepts in computing: past, present, and future. In: *Proceedings of the 16th International Conference on Computing Education Research*, pp. 91–100. ACM (2016)
39. Shaft, T.M., Vessey, I.: The relevance of application domain knowledge: the case of computer program comprehension. *Inf. Syst. Res.* **6**(3), 286–299 (1995)
40. Shopkow, L.: How many sources do i need? *Hist. Teach.* **50**(2), 169–200 (2017)
41. Shopkow, L., Diaz, A., Middendorf, J., Pace, D.: From bottlenecks to epistemology in history: changing the conversation about the teaching of history in colleges and universities. In: *Changing the Conversation about Higher Education*. Rowman & Littlefield Publishing (2013)
42. Timmermans, J., Barnett, J.: The Role of Identifying and Decoding Bottlenecks in the Redesign of Tax Curriculum. In: *Society for Teaching and Learning in Higher Education Conference, Canada* (2013)
43. Verpoorten, D., et al.: Decoding the disciplines — a pilot study at the University of Liege (Belgium). In: *Proceedings of 2nd EuroSoTL Conference*, pp. 263–267 (2017)
44. Weinberg, G.M.: *The Psychology of Computer Programming*. Van Nostrand Reinhold/Litton Educational Publishing (1971)
45. Whalley, J.L., et al.: An Australasian study of reading and comprehension skills in novice programmers, using the bloom and SOLO taxonomies. In: *Proceedings of the 8th Australasian Conference on Computer Science Education*, pp. 243–252 (2006)
46. Wilkinson, A.: Decoding learning in law: collaborative action towards the reshaping of university teaching and learning. *Educ. Media Int.* **51**(2), 124–134 (2014)
47. Willes, K.L.: Data cleaning. In: *The SAGE Encyclopedia of Communication Research Methods*. SAGE (2017)

System Security Education



An Approach to Teaching Secure Programming in the .NET Environment

Sifiso Bangani¹ , Lynn Fletcher¹  , and Johan van Niekerk² 

¹ Department of IT, Nelson Mandela University, Port Elizabeth, South Africa
lynn.fletcher@mandela.ac.za

² Faculty of Computing, Noroff University College, Kristiansand, Norway
johan.vanniekerk@noroff.no

Abstract. The security aspect of software applications is considered as the important aspect that can reflect the ability of a system to prevent data exposures and loss of information. For businesses that rely on software solutions to keep operations running, a failure of a software solution can stop production, interrupt processes, and may lead to data breaches and financial losses. Many software developers are not competent in secure programming. This leads to risks that are caused by vulnerabilities in the application code of software applications. Although various techniques for writing secure code are known, these techniques are rarely fundamental components of a computing curriculum. This paper proposes the teaching of secure programming through a step-by-step approach. Our approach includes the identification of application risks and secure coding practices as they relate to each other and to basic programming concepts. We specifically aim to guide educators on how to teach secure programming in the .Net environment.

Keywords: Programming education · Secure coding practices · Secure programming

1 Introduction

As the world advances in technology by creating new software applications, so does the need to protect these software applications as their vulnerabilities and associated risks also increase. Software applications have become integral to many people as they use them on a day-to-day basis for working with top-secret enterprise intellectual property, sharing personal information, making bank transactions, or sharing pictures with family and friends [12].

Although software plays an important role on a day-to-day basis, it often has associated risks as a result of vulnerabilities in the application layer [29]. The security aspect of software applications is considered as the important aspect that can reflect the ability of a system to prevent data exposures, and loss

Financially supported by the National Research Foundation (NRF), NMU Research Capacity Development (RCD) and BankSETA, South Africa.

of information [18]. Failure to secure software solutions can have more serious effects than just a temporary interruption to a service. For businesses that rely on software solutions to keep operations running, a failure of a software solution can stop production, interrupt processes, and may lead to data breaches and financial losses. The human factor, which includes the programmer, has a major impact on the success and failure of efforts to secure and protect the business, services, and information [17]. According to [30], the main cause of software application failure is human error in application programming, which happens during the coding process.

Software developers are typically equipped with relevant programming knowledge and skills to develop innovative software [25]. However, software developers are rarely equipped with secure programming knowledge and skills from the undergraduate level [19,31]. According to [7], *“students graduating from technical programs such as information technology often do not have the attributes to fill the needs of industry”*. Fundamental programming principles are often introduced to students without an understanding of their security implications. This leads to non-adherence to secure programming [31]. For example, arrays and loops are introduced and explained without the mention of buffer overflows that could occur due to lack of adherence to secure programming.

The purpose of this paper is to help educators to teach secure programming in the .Net environment. Secure programming is an important part of information security education [32], whereby relevant topics of information security ought to be taught to some extent in all modules of the main curriculum throughout all years of study [15]. In this context, the contribution of this paper is five-fold:

1. It identifies relevant application risks in the .Net environment.
2. It identifies secure coding practices to be taught to undergraduate computing students.
3. It determines the basic programming concepts taught in the .Net environment in South African undergraduate computing curricula.
4. It maps the basic programming concepts to relevant application risks.
5. It maps the relevant application risks to the identified secure coding practices.

These mappings help us to understand how to teach secure programming in undergraduate computing curricula. By ‘computing curricula’ we mean university courses that teach programming for Computer Science or BSc IT degrees.

2 Related Work

As much as new software technologies are needed and are being developed, the industry increasingly demands software developers that possess relevant security knowledge, skills and abilities [7]. Advancements in technology also increase the security risks associated with those technologies. This leads to a ‘gap’ of outdated knowledge and skills for industry and academia [14]. In cybersecurity, according to [7], *“although jobs are and will be available, employers find it increasingly difficult to find qualified people to fill them. Students graduating from technical programs such as information technology often do not have the attributes to*

fill the needs of industry". Software security is becoming every company's norm and concern as a result of the rising trend of software application vulnerabilities [12, 14, 26]. This is the driving force behind the demand for software developers with security knowledge and skills. This security skill demand results in industry's need to hire developers experienced in secure programming. These developers must have the knowledge, skills, and abilities of secure programming that enables them to implement security-related solutions.

The security skills scarcity often forces companies to enrol their employees in secure programming certifications such as, IBM's Application Security Analyst Mastery Award, or Microsoft's Software Development Fundamentals course. These certifications are an attempt to make software developers competent in secure programming, as they often lack the required knowledge [9, 24]. However, the knowledge acquired with such certifications is not sufficient to be productive in secure programming without the necessary skills, as there should be a balance between knowledge and skills [28]. Secure programming certifications and training focus on two primary factors, namely: awareness of a specific security threat, and having adequate training in the use of the security counter-measure to such a threat [1]. However, these certifications and training do not guarantee a change in human behaviour [1, 17], as human behaviour requires more than just awareness of a specific security threat. For software developers to be competent in secure programming, they must be trained on the requisite skills of secure programming at an undergraduate level.

Producing competent software developers should therefore begin in universities and colleges where students are being educated in understanding and applying learned concepts and the ability to work in a team environment [6, 14]. Universities are responsible for providing a hands-on teaching approach for undergraduate students, which includes lectures, computer laboratory practicals, and experiments [28, 29]. The fundamentals of computing are introduced to students at university level. The learning outcomes of university curricula are used to show what the students will know and be able to demonstrate after the completion of that course [14]. They are key to the shift of focus in education from a paradigm of providing instructions to a paradigm of producing learning [3].

Various computing curricula guidelines, such as the Association for Computing Machinery's (ACM), state that Information Assurance and Security (IAS) belong to an advanced level of a four-year computing program. However, many students in three-year computing courses graduate and leave university before completing a fourth year of study [15]. This implies that many undergraduate computing students are often not adequately exposed to secure programming. Since students can only apply what they have been taught, the behaviour of a student in a certain area such as secure programming can be improved by providing students with the requisite knowledge [20]. This can be done through software security education in computing at universities.

Software security is the idea implemented to protect software to ensure that it functions correctly under malicious attacks [16]. Furthermore: *"Software security is about building secure software: designing software to be secure, making sure*

that software is secure, and educating software developers, architects, and users about how to build secure things" [16]. Software security is not simply implemented by installing an anti-virus software to a computer or mobile device, as hackers steal or get access to top secret enterprise information, or even damage the behaviour of software applications. Hackers can damage software through embedding malicious software or scripts in the code. Software applications without proper security built-in can be vulnerable to various computer attacks such as Cross-Site Scripting, SQL Injections, Session Hijacking, Cross-Site Request Forgery, and Denial of Service attacks [29]. The only way to avoid such attacks is by practising good secure programming techniques [2,30].

Secure programming is the manner of writing code to minimise software security vulnerabilities, as many problems faced by users nowadays are caused by vulnerabilities resulting from flaws in application code. Various techniques for writing secure code are known [23,27,31]. Although these techniques exist, they are rarely fundamental components of a computing curriculum, but rather treated as secondary topics that are briefly discussed in programming courses [31]. To maintain security in software applications, students must have the necessary skills and knowledge. According to [5], *"the ability to write secure code should be as fundamental to a university computer science undergraduate as basic literacy"*. The following section briefly describes computing education in the South African context.

3 Computing Education in the South African Context

Institutions of higher learning in South Africa are divided into public and private universities [11]. In this paper our focus is on the public universities. South African public universities are divided into three categories, namely: traditional universities, universities of technology, and comprehensive universities. The South African public universities are overseen by the government's Department of Higher Education and Training (DHET) which is responsible for post-school education and training.

South African universities offer three-to-four-year degree qualifications depending on the type of university [10]. Comprehensive universities and universities of technology offer three-year diploma qualifications for workplace readiness [15]. For a student in such universities to obtain a degree, the student would be required to advance the diploma qualification by completing a fourth year of study. Such a fourth year of study is considered to be 'advanced', wherein students can be introduced to higher-level topics [14]. In the case of programming qualifications, the fourth year of study would typically include an introduction to security and secure programming basics [14,15]. Traditional universities with three-year degree qualification teach programming basics in the undergraduate level. In traditional universities the fourth year of study (a.k.a. 'honours' in South Africa) is also considered to be an 'advanced' level wherein students are introduced to higher-level topics.

South African universities offer semester courses as well as year courses. Semester courses are usually carried out over a period of six months, whereas year

courses are carried out throughout the academic year [11]. Both types of courses in various universities offer fundamentals of programming. However, secure coding practices are rarely explicitly taught to undergraduate students. They are rather treated as secondary topics that are briefly discussed in those programming courses [31]. Examples of such courses include: Programming Fundamentals, Computing Fundamentals, Development of Software, Applications Development, Mobile Computing, Technical Programming, or Web Systems.

In this paper the focus is on applications development in the .Net environment, since most South African universities teach programming with it (whereby Microsoft promotes free product usage by university students). Nonetheless our proposed approach to teaching secure programming practices in undergraduate computing curricula is suitable for other development environments and frameworks as well.

4 Research Approach

A preliminary investigation and content analysis were conducted to determine whether South African universities incorporate secure programming in their undergraduate computing curricula. This preliminary investigation was conducted on South African universities through a thematic content analysis by reviewing the Prospectus or Study-Guides of various universities, where relevant themes and topics relating to secure programming were sought. The purpose of the investigation was to determine whether secure programming is already included in the teaching of programming concepts, as writing secure code ought to be fundamental to an undergraduate computing student [5]. A content analysis is typically conducted to make replicable and valid inferences from texts and data [13,20]. For this paper, content analysis was applied w.r.t. the above-mentioned matters.

Figure 1 sketches our research process which led to a step-by-step approach for teaching secure programming in undergraduate computing curricula with the following six steps; (for additional recommendations see [19]).

- Step 1:** Identification of relevant application risks in the .Net environment that are important for teaching secure programming.
- Step 2:** Identification of secure coding practices that should be taught to computing students as requisite knowledge for secure programming.
- Step 3:** Identification of basic programming concepts typically taught to undergraduate students (in the .Net environment).
- Step 4:** Mapping application risks to programming concepts to demonstrate the need for teaching application risks along with programming concepts.
- Step 5:** Mapping basic programming concepts to identified secure coding practices in order to highlight the need for, and relevance of, integrating secure coding practices to programming concepts taught.
- Step 6:** Mapping application risks to identified secure coding practices in order to show the relationship between application risks and secure coding practices to highlight the importance of secure programming.

Each step is described in further detail in the following sub-sections.

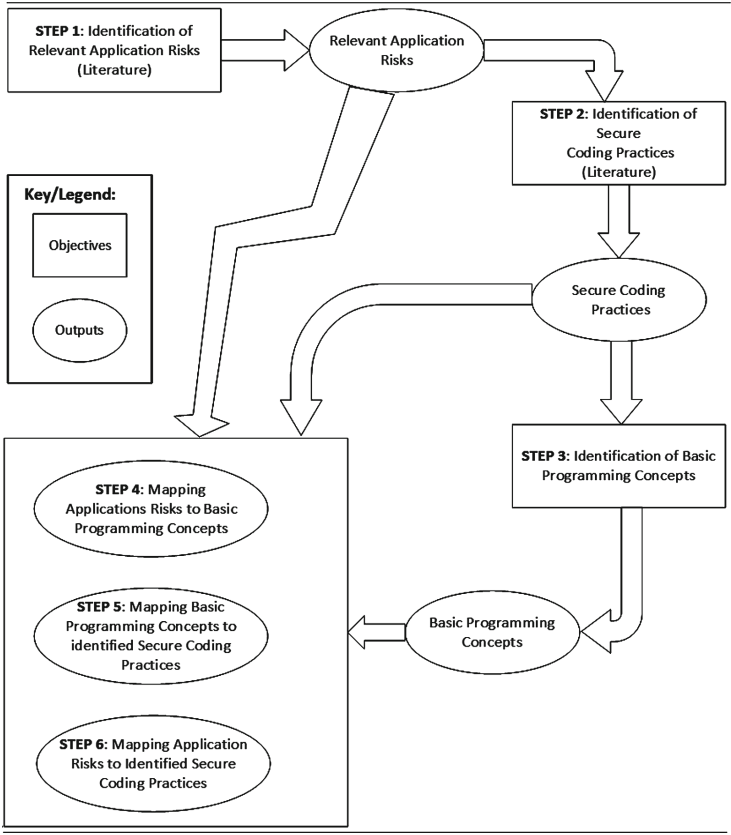


Fig. 1. Research process

4.1 Step 1: Identification of Relevant Application Risks (ARs)

An initial literature review identified application risks that can affect software applications developed in the .Net environment. The Open Web Application Security Project (OWASP) was used as a source of software application security guidance. OWASP is an international not-for-profit group dedicated to helping organisations to develop, purchase, and maintain good software applications [22]. OWASP is known for providing free documents of application risks, including a ‘Top 10 Application Risks’ document for awareness in *web application security* [7]. This document represents a broad consensus about the most critical security risks to web applications [23]. Although this OWASP list is mostly relevant to web applications, it can also be used for other software applications during application development, testing, and maintenance. The examples provided in this paper relate to web applications as they are often deemed the most vulnerable software applications.

Table 1. OWASP top 10 application risks for the year 2017

AR 1	Injection
AR 2	Broken authentication and session management
AR 3	Sensitive data exposure
AR 4	XML external entities
AR 5	Broken access control
AR 6	Security misconfiguration
AR 7	Cross-site scripting
AR 8	Insecure deserialisation
AR 9	Use of components with known vulnerabilities
AR 10	Insufficient logging and monitoring

Table 1 shows the OWASP Top 10 Application Risks (ARs) ordered by their severity. This list can also be used in the development of other software solutions that are not .Net-based, to guide and test for well-known vulnerabilities, as these application risks can affect most applications regardless of the development environment. In the identification of these application risks, the SANS Top 25 Programming Errors list [8] was used to compare the current application risks to well-known programming errors, to determine the extent to which the errors could cause the risks listed in OWASP’s list of Top 10 Application Risks. Some errors in the SANS Top 25 Errors list are no longer critical, as there have been changes in the security of development platforms and frameworks. This also resulted in the change in OWASP’s list of the Top 10 Application Risks, with cross-site scripting dropping from rank 2 in the list of 2013 to rank 7 in the list of 2017 [8, 23].

4.2 Step 2: Identification of Secure Coding Practices (SPs)

To identify the secure coding practices, a literature review identified principles, techniques, and practices of secure programming. The literature review of fundamental secure coding practices was conducted to understand what software developers need to be competent in w.r.t. secure programming [6].

The Secure Coding Practices Checklist recommended by OWASP was used for the identification of secure coding practices. The OWASP Secure Coding Practices Checklist can be used to mitigate most common software application vulnerabilities [22]. This checklist addresses the application risks listed in Table 1 and is used later in this paper to map with application risks and basic programming concepts. Table 2 shows OWASP’s Secure Coding Practices Checklist: the encoding identifier is SP followed by its rank number in the list.

In the ongoing investigation of secure coding practices to be taught to undergraduate students, the concept map by the University of California’s Davis Secure Programming Clinic [4] was reviewed for the identification and classification of programming practices. The identification and classification of the

Table 2. OWASP secure coding practices checklist

SP 1	Input validation
SP 2	Output encoding
SP 3	Authentication and password management
SP 4	Session management
SP 5	Access control
SP 6	Cryptographic practices
SP 7	Error handling and logging
SP 8	Communication security
SP 9	System configuration
SP 10	Database security
SP 11	File management
SP 12	Memory management
SP 13	General coding practices

secure coding practices was supported by the OWASP Secure Coding Practices Checklist [22]. This was done to assess the validity of the guidelines and principles in the current secure programming clinic.

4.3 Step 3: Identification of Basic Programming Concepts (PCs)

Having identified the secure coding practices, the globally published curricula guidelines for undergraduate computing programs were reviewed to determine the extent to which secure programming should be addressed in Computer Science (CS) and Information Technology (IT) qualifications. The focus was on the ACM curricula documents, as the ACM updates and adapts curricula recommendations quickly to the rapidly changing landscape of computer technology. Although the ACM curricula guidelines mention security as part of computing curricula, the guideline documents for CS and IT do not have adequate guidance on how secure programming can be taught to enable a graduate software developer to be competent in secure programming. The key to educating and training software developers is typically found in the Prospectus or Study-Guides pertaining to each university [14, 28].

To understand the state of programming in the undergraduate level, a thematic content analysis was conducted on undergraduate computing curricula in South Africa. The content analysis was done to determine basic programming concepts taught in the .Net environment, across different public universities in South Africa. The Prospectus and Study-Guides that are available on the universities' websites were used for this purpose.

Table 3 provides a list of basic programming concepts (PCs) that are typically taught in South African universities. The list does not provide any 'correct order' in which the concepts are taught; it only outlines the fundamentals of

programming for beginners developing in the .Net environment. Each PC item in the list is followed by its position number. Although PC8 and PC9 are not ‘concepts’ in the strict sense of the term, they were deemed essential enough by most universities to be taught to beginners developing in the .NET environment.

Table 3. Basic programming concepts for beginners in the .Net environment

PC 1	Variable declaration (with data types)
PC 2	Conditional structures
PC 3	Arrays (including search and update)
PC 4	Collections (arrays, lists, stacks, queues)
PC 5	Loops (while, for, until)
PC 6	Database connection
PC 7	File operations
PC 8	Basic HTML and XML
PC 9	JavaScript
PC 10	Web control
PC 11	Data binding
PC 12	Error handling
PC 13	Validation
PC 14	State management
PC 15	Master pages and layouts

4.4 Step 4: Mapping Application Risks (ARs) to Basic Programming Concepts (PCs)

After consolidating findings about what programming concepts should be included when teaching secure programming in South African universities, we created mapping links of how application risks can be taught when teaching programming concepts. The purpose of these mapping links is to demonstrate the need for, and relevance of, teaching application risks along with programming concepts. The mapping links in the content analysis results were given an impact value (**I**). **I** is based on how many times a programming concept (PC) was linked to an application risk (AR). **I** can also be seen as a way of prioritising important links. Figure 2 shows the mapping links between the identified programming concepts and the OWASP Top 10 Application Risks.

The mapping of programming concepts to application risks shows the relationship that the programming concept has directly to the application risk. A programming concept can have a number of application risks associated with it, and an application risk can occur due to poorly written programming concepts. A programming concept that links with many application risks receives a high impact value (**I**), where an impact value of 4 and above would mean that the link needs special attention. Therefore, educators of programming courses could

Programming Concept	OWASP Top 10 Application Risks 2017										(<i>I</i>)
	AR1	AR2	AR3	AR4	AR5	AR6	AR7	AR8	AR9	AR10	
PC1: Variable Declaration		X									1
PC2: Conditional Structures *	X	X			X			X			4
PC3: Arrays	X										1
PC4: Collections	X		X								2
PC5: Loops	X										1
PC6: Database Connection *	X	X	X	X	X						5
PC7: File Operations	X			X		X					3
PC8: Basic HTML & XML *	X			X		X	X		X		5
PC9: JavaScript		X					X		X		3
PC10: Web Controls *	X					X	X		X		4
PC11: Data Binding	X										1
PC12: Error Handling *	X	X	X		X	X				X	6
PC13: Validation *	X	X	X		X		X	X			6
PC14: State Management		X									1
PC15: Master Pages & Layouts		X			X						2
Impact (<i>I</i>)	11	8	4	3	5	4	4	2	3	1	(<i>I</i>)

Fig. 2. Mapping of basic programming concepts to OWASP Top 10 application risks

prioritise the lecture time taken for each link according to its impact value. In this paper, the programming concepts with a high impact value will be used to highlight the importance of considering application risks when teaching programming.

For example, the programming concept Error Handling (PC12) links to many application risks and, thus, it received a high impact value (**I**) of 6. Error handling is the last defence in a software application when written code statements do not execute as expected [22,30]. To educate students on how to program securely, the associated application risks must be taught after the introduction of the programming concept. The introduction of these application risks should begin from the first year of study, and should be taught in parallel with programming concepts as they occur in the syllabus. In an attempt to ensure that students adhere to secure programming and avoid these application risks, students must implement a means that recovers from errors (e.g. try-catch) [31].

Similarly, the programming concept Validation (PC13) received a high impact value (**I**) of 6 which shows its importance to software applications. Applications without proper validation of data can be vulnerable to various applications risks [23]. Students must therefore be encouraged to always use input validation to avoid the application risks such as Injection (AR1) associated with Validation (PC13) in Fig. 2 [9]. Encouraging students to do Validation (PC13) would require educators to examine the students’ adherence by setting laboratory practicals that require input validation. Students should be assessed and their work graded by reviewing the code they develop.

In Fig. 2, Injection (AR1) is the first in the OWASP Top 10 Application Risks, which shows how critical this risk is to software applications [23]. This application risk should be introduced and taught in parallel with the associated programming concepts to avoid this risk from occurring. To avoid Injections (AR1), students must be taught how they relate to each of the associated programming concepts (i.e. PC2, PC6, PC8, PC10, and PC12–13).

4.5 Step 5: Mapping Basic Programming Concepts (PCs) to Secure Coding Practices (SPs)

After understanding the application risks that must be taught to undergraduate computing students, we created mapping links of how secure coding practices can be taught in basic programming concepts. The purpose of the mapping links is to demonstrate the need for, and relevance of, integrating secure coding practices to basic programming concepts. The mapping links identified in the content analysis results were given an impact value (**I**); see Fig. 3. **I** is based on how many times a programming concept (PC) was linked to a secure coding practice (SP) horizontally according to the programming concept (PC), and how many times a secure coding practice (SP) was linked to a programming concept (PC) vertically. **I** can thus be seen as a way of prioritising important links that need special attention.

Programming Concept	OWASP Secure Coding Practices Checklist													I
	SP1	SP2	SP3	SP4	SP5	SP6	SP7	SP8	SP9	SP10	SP11	SP12	SP13	
PC1: Variable Declaration	x			x									x	3
PC2: Conditional Structures	x		x	x	x		x	x					x	7
PC3: Arrays	x													1
PC4: Collections	x						x					x		3
PC5: Loops												x		1
PC6: Database Connection	x		x	x	x					x				5
PC7: File Operations	x	x					x		x		x	x		6
PC8: Basic HTML & XML		x	x						x					3
PC9: JavaScript	x										x		x	3
PC10: Web Controls	x						x						x	3
PC11: Data Binding							x			x		x	x	4
PC12: Error Handling	x	x	x	x	x	x	x		x	x	x	x	x	11
PC13: Validation	x	x	x	x	x	x	x	x	x	x	x	x	x	13
PC14: State Management	x				x		x							2
PC15: Master Pages & Layouts									x				x	2
Impact I	11	4	5	5	5	2	8	2	5	4	4	6	7	I

Fig. 3. Mapping of basic programming concepts to secure coding practices

The mapping link between programming concepts and secure coding practices shows a direct relationship between them. A programming concept can have a number of secure coding practices that can be associated with it, and a secure coding practice can be applied to a number of programming concepts. A programming concept that links with many secure coding practices receives a high impact value (**I**), whereby an impact value of ≥ 4 indicates that such a link ought to be given special attention.

Figure 3 shows that the programming concepts Error Handling (PC12) and Validation (PC13) in this mapping are most important, as they received the highest impact values (**I**) of 11 and 13 respectively. (Similarly, in Fig. 2, PC12 and PC13 achieved high impact values of 6.) Most application failures are a result of lack of Error Handling (PC12) and poor Validation (PC13). For Input Validation (SP1) to work effectively, it is mostly used with Conditional Structures (PC2) to avoid errors that might occur due to a lack of Error Handling (PC12) and Validation (PC13). When Input Validation (SP1) is not properly implemented, an application can be vulnerable to many application risks as shown in Fig. 2. When educators teach these programming concepts, they should therefore pay specific

attention to the impact caused by the association, and try to keep a balance between the programming concept and its associated secure coding practices.

4.6 Step 6: Mapping Application Risks (ARs) to Identified Secure Coding Practices (SPs)

After understanding the secure coding practices that must be taught to undergraduate computing students, we created mapping links that show the relationships between application risks and secure coding practices. As above, the mapping links in the content analysis results were given an impact value (**I**). Here, **I** is based on how many times an application risk (AR) was linked to a secure coding practice (SP) horizontally, and how many times a secure coding practice (SP) was linked to an application risk (AR) vertically; see Fig. 4. Again, **I** can be seen as a way of prioritising important links that need special attention.

Application Risk	Secure Coding Practices													(<i>I</i>)
	SP1	SP2	SP3	SP4	SP5	SP6	SP7	SP8	SP9	SP10	SP11	SP12	SP13	
AR1: Injection	x	x					x			x	x	x	x	6
AR2: Broken Authentication	x		x	x	x	x	x		x				x	8
AR3: Sensitive Data Exposure						x			x	x	x			5
AR4: XML External Entities		x	x	x					x					4
AR5: Broken Access Control	x		x	x	x	x	x							6
AR6: Security Misconfiguration		x		x			x	x	x	x	x	x	x	9
AR7: Cross-Site Scripting	x	x					x		x		x			5
AR8: Insecure Deserialization	x	x					x		x					4
AR9: Using components with known vulnerabilities		x											x	2
AR10: Insufficient Logging and Monitoring							x							1
Impact (<i>I</i>)	5	6	3	4	4	2	7	1	6	3	4	2	4	(<i>I</i>)

Fig. 4. Mapping of identified application risks to secure coding practices

The mapping links between application risks and secure coding practices show a direct relationship between them. An application risk can have a number of secure coding practices that address it, and a secure coding practice can be applied to mitigate a number of application risks. An application risk that links with many secure coding practices receives a high impact value (**I**), where (**I**) of ≥ 4 would mean that such a link ought to receive special attention. Therefore, educators must not teach application risks and secure coding practices in isolation; the secure coding practices in Table 2 are used to prevent or mitigate the application risks in Table 1. Broken Authentication (AR2) in Fig. 4 links with many secure coding practices; thus it receives a high impact value (**I**) of 8. Software applications without a properly structured authentication mechanism can be vulnerable to privilege escalation [22,30]. The application risks Broken Authentication (AR2) and secure coding practice Authentication and Password Management (SP3) provide an example that can be used to teach students not to hard-code passwords, nor to leave plaintext passwords in the configuration files, as that can enable attackers to bypass access controls [30]. Error Handling and Logging (SP7) received a high impact value (**I**) of 7, which shows the importance that error handling and logging has in avoiding application risks such as

Sensitive Data Exposure (AR3). When error handling and logging is properly used in an application, default errors that show critical information such as server details are avoided by showing a custom error created by the programmer [21,30]. To avoid Security Misconfiguration (AR6), students must be taught to avoid insecure default configurations and verbose error messages containing sensitive information. For ASP.Net applications, students can avoid Security Misconfiguration (AR6) by being taught to properly configure the `.config` file in the solution. A typical example of configuring the `.config` file would be to enable `customErrors`, so that default error messages will not be displayed.

5 Discussion and Conclusion

For (under)graduate software developers to be competent in secure software development they should be equipped with relevant and necessary secure programming knowledge in the curriculum. The literature shows that secure coding practices and techniques already exist [23,27,31]. However, they are rarely used as fundamental components of computing curricula, but are rather treated as secondary topics which are merely briefly discussed in programming courses [31].

Universities are responsible for educating undergraduate computing students where fundamentals of computing are introduced to students who are guided through practical classes in the computer laboratories [28]. Although many universities teach programming, often only little attention is given to secure programming, which results in incompetent undergraduate software developers. A university's Prospectus or Study-Guides are key to teaching undergraduate students, as these documents show what the student will know and be able to do apply after completion of the course.

Computing curricula reports such as the various ACM curricula guidelines recommend the teaching of secure programming in undergraduate computing courses. However, these guidelines do not provide adequate guidance on how secure programming can be integrated into the curriculum to enable a graduate software developer to be competent in secure programming.

The step-by-step approach proposed in this paper can be used at various levels for preparing a computing curriculum. Our approach can be used in setting up the Prospectus and Study-Guides, to ensure that relevant application risks and secure coding practices are appropriately considered. The steps proposed by this paper go hand-in-hand and cannot be addressed in isolation, as isolating these steps may lead to vulnerabilities that can affect a software application.

In addition, the mappings presented in this paper show the relationships between the programming concepts taught to undergraduate students with the identified application risks and secure coding practices. These mappings serve as a guide for how the application risks can be addressed by considering secure coding practices relating to basic programming concepts. Secure coding practices must be explicitly taught in undergraduate computing curricula to ensure that students will be competent in secure software development.

This paper proposes that secure coding practices be taught throughout the undergraduate programming modules, from the first year of study throughout

to the final year of study. This approach would not only impact the competence of graduate software developers, but it would positively influence the security of software applications developed by these graduate software developers for the benefits of society.

The main limitation of this paper is that the approach and the mappings suggested in this paper have not yet been thoroughly validated. This will be part of future research as well as the actual implementation of this approach at various universities in South Africa. In the next-following paper of this CCIS volume we address the ‘pervasive integration’ of secure coding principles into the entire computer science curriculum [19].

References

1. Aytes, K., Conolly, T.: A research model for investigating human behavior related to computer security. In: Proceedings of the 9th Americas Conference on Information Systems, pp. 1–6 (2003)
2. Aziz, N.A., Shamsuddin, S.N.Z., Hassan, N.A.: Inculcating secure coding for beginners. In: Proceedings of the ICIC International Conference on Informatics and Computing, pp. 164–168. IEEE (2016)
3. Barr, R.B., Tagg, J.: From teaching to learning - a new paradigm for undergraduate education. *Change Mag. High. Learn.* **27**(6), 12–26 (2012)
4. Bishop, M.: A clinic for secure programming. *IEEE Secur. Priv. Mag.* **8**(2), 54–56 (2010)
5. Bishop, M., Frincke, D.A.: Teaching secure programming. *IEEE Secur. Priv.* **3**(5), 54–56 (2005)
6. Buoncristiani, M., Buoncristiani, P.: How People Learn (2014). <https://doi.org/10.4135/9781483387772.n2>
7. Burley, D., Bishop, M., Buck, S., Ekstrom, J., Fitcher, L., Gibson, D.: Joint Task Force on Cybersecurity Education (2017). <http://www.csec2017.org/>
8. Christey, S., Martin, B.: CWE-2011 CWE/SANS Top 25 Most Dangerous Software Errors (2011). <http://cwe.mitre.org/top25/#CWE-209>
9. Cotler, J., College, S., Mathews, L., College, S., Hunsinger, S.: Information systems applied research 2015 AITP education special interest group (EDSIG) board of directors. *Inf. Syst. Appl. Res.* **8**(1), 1–65 (2015)
10. Department of Education: Creating Comprehensive Universities in South Africa: a Concept Document. Rep. of South Africa (2004)
11. Department of Education: Regulations for the Registration of Higher Education. Rep. of South Africa (1997)
12. Hoekstra, M., Lal, R., Pappachan, P., Phegade, V., del Cuvillo, J.: Using innovative instructions to create trustworthy software solutions. In: Proceedings of the HASP 2013 2nd International Workshop on Hardware and Architectural Support for Security and Privacy, p. 1 (2013)
13. Krippendorff, K.: Content Analysis: An Introduction to its Methodology (1985). <https://doi.org/10.1103/PhysRevB.31.3460>
14. Lunt, B.M., Ekstrom, J.J., Lawson, E.: Curriculum guidelines for undergraduate degree programs in information technology (2008)
15. Mabece, T., Fitcher, L., Thomson, K.L.: Towards using pervasive information security education to influence information security behaviour in undergraduate computing graduates. In: Proceedings of the CONFIRM 2016, p. 14 (2016)

16. McGraw, G.: Software security. *EEE Secur. Priv.* **2**(2), 80–83 (2004)
17. Metalidou, E., Marinagi, C., Trivellas, P., Eberhagen, N., Skourlas, C., Giannakopoulos, G.: The human factor of information security: unintentional damage perspective. *Procedia Soc. Behav. Sci.* **147**, 424–428 (2014)
18. Mumtaz, H., Alshayeb, M., Mahmood, S., Niazi, M.: An empirical study to improve software security through the application of code refactoring. *Inf. Softw. Technol.* **96**, 112–125 (2018)
19. Ngwenya, S., Fitcher, L.: A framework for integrating secure coding principles into undergraduate programming curricula. In: Tait, B., et al. (eds.) *SACLA 2019. CCIS*, vol. 1136, pp. 50–63 (2020)
20. van Niekerk, J.F., von Solms, R.: Information security culture: a management perspective. *Comput. Secur.* **29**(4), 476–486 (2010)
21. OWASP: Secure Coding Practices Checklist (2016). <https://www.owasp.org/>
22. OWASP: Secure Coding Practices Quick Reference Guide (2010)
23. OWASP: Top 10 2017: The Ten Most Critical Web Application Security Risks (2017). <https://www.owasp.org/>
24. Perrone, L.F., Aburdene, M., Meng, X.: Approaches to undergraduate instruction in computer security. In: *Proceedings of the ASEE Annual Conference and Exhibition*, pp. 651–663 (2005)
25. Rajlich, V.: Teaching developer skills in the first software engineering course. In: *Proceedings of the ICSE*, pp. 1109–1116 (2013)
26. Ramachandran, M.: Software security requirements management as an emerging cloud computing service. *Int. J. Inf. Manag.* **36**(4), 580–590 (2016)
27. Singhal, A., Winograd, T., Scarfone, K.: Guide to secure web services. NIST Special Publication 800–95 (2007)
28. The Joint Task Force on Computing Curricula: Information Technology Curricula 2017: Curriculum Guidelines for Baccalaureate Degree Programs in Information Technology. ACM/IEEE (2017)
29. Uskov, A.V.: Hands-on teaching of software and web applications security. In: *Proceedings of the IEDEC 3rd Interdisciplinary Engineering Design Education Conference*, pp. 71–78 (2013)
30. Veracode: State of Software Security (2017)
31. Whitney, M., Lipford, H.R., Chu, B., Thomas, T.: Embedding secure coding instruction into the IDE: complementing early and intermediate CS courses with ESIDE. *J. Educ. Comput. Res.* **56**(3), 415–438 (2018)
32. Wu, D., Fulmer, J., Johnson, S.: Teaching information security with virtual laboratories. In: Carroll, J.M. (ed.) *Innovative Practices in Teaching Information Sciences and Technology*, pp. 179–192. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-03656-4_16



A Framework for Integrating Secure Coding Principles into Undergraduate Programming Curricula

Sandile Ngwenya[✉] and Lynn Futcher[✉]

Department of IT, Nelson Mandela University, Port Elizabeth, South Africa
lynn.futcher@mandela.ac.za

Abstract. The rise of the use of the internet has led to significant growth in software applications for conducting business, entertainment and socialising, which in turn has led to a higher rate of attacks on software applications. This problem has led to industry requiring software developers skilled in developing software in a secure manner. The problem that industry faces is that many software development graduates do not have the requisite knowledge in secure programming. Academia should thus address these needs of industry by integrating secure coding principles into undergraduate programming curricula. In South Africa, however, this is often not formally done. This paper suggests some secure coding principles that could be integrated into programming curricula, together with various integration approaches and related challenges. It presents a framework for integrating secure coding principles into undergraduate programming curricula to ensure the formal planning and ‘buy-in’ of academic staff at all levels. The purpose of the framework is to guide computing faculties about ‘what’ secure coding principles to teach and ‘where’ to teach them.

Keywords: Undergraduate curricula · Secure coding principles · Secure programming

1 Introduction

Secure coding, also known as secure programming, is the practice of developing software programs that are safe from attacks [4]. This definition points us to the practical aspect of the application of security to software. Over the past decade, the software development industry’s focus has grown to include security. This change in focus signifies the need for software developers who can program secure software applications. This need has resulted in a demand for continuing secure coding education as it was already stated over a decade ago [12] and has been further compounded due to the increase in the number of application vulnerabilities prevalent in software applications today.

Financially supported by the National Research Foundation (NRF), South Africa, and the NMU Research Capacity Development (RCD) programme.

According to [9], many undergraduate programming courses teach some elements of secure coding, such as good program structure, basic input validation, checking bounds for array references, and checking that pointers are non-null. However, many programming courses tend to focus more on the skills required for developing functional and user-friendly applications than on the security aspects. Without a conscious learning effort from both lecturers and students towards secure coding, software applications developed by many programming graduates will remain prone to vulnerabilities and susceptible to attacks.

According to [22], teaching secure coding has never been more important than nowadays, such that the need for the formal inclusion of secure coding principles into undergraduate programming curricula must be effected. These secure coding principles should be assessed both theoretically and practically. The support for the inclusion of security in undergraduate programming curricula by various ACM curricula documents [6, 19, 23] serves to show that attention to secure coding is mandatory for the education of computing students. We should ensure they have the ability to develop software applications that can perform the vital function of consistently securing critical data and information.

In order to address the problem of the lack of formal inclusion of secure coding principles into undergraduate programming curricula, this paper—which can be read as a continuation of our foregoing paper [5] in this CCIS volume—firstly highlights the need for teaching secure coding principles in undergraduate computing curricula (as determined by various ACM curricula documents). Secondly, it presents some secure coding principles to be considered in the teaching of secure coding. Furthermore, it describes the various challenges and approaches to integrating secure coding principles into programming curricula. Finally, it proposes a three-phased approach to integrate secure coding principles into undergraduate programming curricula in the form of a framework.

2 Related Work

The Association for Computing Machinery (ACM) has an ongoing education initiative in which it produces and updates curricular recommendations in computer science, computer engineering, information systems, information technology, and software engineering, that are trusted resources utilised by computing faculties globally [1, 6, 19, 23]. As the field of IT continues to change and grow rapidly, IT curricula must follow suit. Inculcating modern skills into the IT curriculum prepares students for professional practice upon graduation [19]. Furthermore, [19] states that it is vital to include professional preparedness into IT curricula because graduates of IT programs will be faced with real-world problems after graduation. In this context the ‘Cyber Principles’ domain is considered essential [19].

In addition, [19] specifically includes a *“focus on implementation, operation, analysis, and testing of the security of computing technologies”* as part of its scope. As part of the ‘Integrated Systems Technology’ domain it requires undergraduate programming students to be able to *“illustrate the goals of secure coding and show how to use these goals as guideposts in dealing with preventing buffer overflow, wrapper code, and securing method access”*. This document thus confirms that secure coding education must be part of undergraduate programming curricula.

Similarly, [23] describes an Information Assurance and Security Knowledge area for which it demands security education in undergraduate Computer Science curricula. The document states: “*Information assurance and security as a domain is the set of controls and processes... intended to protect and defend information and information systems by ensuring their confidentiality, integrity, and availability, and by providing for authentication and non-repudiation*” [23]. The mention of the technical aspect of security suggests that the practical aspects of security such as secure coding are relevant in undergraduate programming curricula. Security education therefore includes all efforts to prepare graduates with the needed knowledge, skills, and abilities to develop information systems and attest to the security of processes and data [23].

Also according to [6] developers must ensure that their software is designed to meet security requirements. Again this implies an appropriate education at undergraduate level.

3 Secure Coding Principles

Secure coding aims at a software product’s robustness against accidental or malicious unexpected behaviour that causes a problem [8]. Coding responsibly means knowing how to develop secure code which is essential for the implementation of modern software systems [2]. As mentioned above it is important that undergraduate students learn how to code responsibly by being taught the theoretical and practical aspects of various secure coding principles.

The implementation of secure coding principles is key in the development of software applications that adhere to security requirements. The list given below provides examples of secure coding principles that can be taught in undergraduate programming curricula as identified by OWASP [18]. The same principles also occur in the Software Security Knowledge area of [6]. Note that this list is not all-encompassing, but contains some fundamental secure coding principles that can be formally included in undergraduate programming curricula.

1. **Input Validation** refers to the process of ensuring that data that is entered as input in applications is filtered and protected against ‘unclean’ data.
2. **Authentication and Password Management** is defined by [7] as the process of a server deciding whether a user should be allowed to login or not. Authentication controls must be validated on a trusted system (e.g. a server).
3. **Session Management** tracks the activity of users as they interact with a website across sessions. Its most widespread use is the login functionality, but it is also used to track other types of interactions users may have with a website [24].
4. **Access Control** ensures that restricted areas of a software system are only accessed by authorised users. A user would have to provide credentials to be granted access.
5. **Cryptographic Practices** involve the conversion of data into a format that is unreadable to users that are not authorised to do so [10].

6. **Error Handling and Logging** entails the recovery of an application from an error condition using recovery responses. Error handling anticipates the possibility of error conditions, detects errors, and provides a resolution of an error to maintain the execution of an application.
7. **Data Protection** aims to keep private data only available for business purposes or to maintain data privacy rights [20].
8. **Database Security:** Information stored in today's databases is highly valuable and confidential. As such, secure coding must play an active role in the protection of that information from unauthorised access to it [15].
9. **File Management:** File types include text files, data files, binary, or graphic files. File upload controls should support anti-malware and anti-virus capabilities to avoid the upload of files that can compromise the application or database where they would get to be saved [3].

Table 1. Secure coding principles and related content according to [18]

SCP	RELATED CONTENT
1	Input validation should be processed on a trusted system (e.g. a server); input that fails validation should be rejected; expected data types must be validated; the data range of input must be validated
2	All authentication should be processed on a trusted system (e.g. a server); passwords stored on a database must be encrypted; password complexity must be enforced; minimum password length must be validated
3	Logout functionality should be protected by authorisation across all pages where it is used in a website; a session must be configured to expire within a reasonably short amount of time; Session identifiers must not be exposed such as on URL headers
4	Access controls must fail securely if they fail; only authorised users may be granted access to services
5	All cryptographic functions used should be processed on a trusted system (e.g. a server); highly sensitive data must be encrypted
6	Do not disclose sensitive information in error responses, including system details, session identifiers or account information; feedback must be in the form of generic error messages, and custom error pages must be used
7	Feedback must be in the form of generic error messages, and custom error pages must be used; URL headers must not include sensitive information
8	Parametrised queries must be used for the interaction between the application and a database; connection strings should be stored and encrypted separately in a configuration file on a trusted system; they must not be hard-coded
9	Files should never be uploaded without authentication; only allow the upload of file types that are required for business purposes

These nine secure coding principles (SCP) are listed again in Table 1 together with the related topic-content that can be taught in programming curricula. Programming lecturers can use the related content in Table 1 as a basis for determining the learning outcomes that can be achieved for each secure coding principle. This content should be aligned with the relevant undergraduate programming curricula as seen fit by the educators of such computing courses.

4 Challenges and Approaches to Integrating Secure Coding Principles into Programming Curricula

There are several challenges that affect undergraduate programming curricula in integrating learning outcomes that are specifically aimed at teaching secure coding principles. Although there might be a genuine desire and interest in teaching secure coding [8], the reality is that it is not an easy task to accomplish. Three challenges that hinder progress in the teaching of secure coding include [8]:

1. Lack of room in the already ‘crammed’ software development curricula;
2. Focus on introductory software development matters;
3. Teaching in a manner that does not promote the consequent application of already learnt programming techniques.

The first challenge is due to the notion that software development courses that include secure coding principles must be separate courses. This is an assumption as introductory coding courses already teach some of the basics of secure coding such as validating inputs and the handling of exceptions [22]. There are various ways of adding secure coding principles to undergraduate programming curricula. The common approach would be to add lectures and practical classes that include the teaching of secure coding [25].

The second challenge stems from the primary focus on introductory undergraduate programming courses which is to teach programming logic—whereby many students already fail [14]. For many computing faculties, the problem is knowing where to infuse the secure coding principles within the sequence of the primary focus. Undergraduate programming courses mainly focus on algorithmic and programming language issues rather than secure coding. The norm is that undergraduate students do not need to know the secure coding concepts in detail [8]. This leads to students not being sufficiently equipped.

The third challenge speaks to the practical application of what students learn when they are taught how to code and how their work is graded. It is assumed that students can be introduced to basic secure coding techniques and that they then apply the principles of secure coding by themselves. However, students tend to focus their preparations rather narrowly according to what they are taught and assessed on. The result is that students do not apply the principles of secure coding, as most often the grading only focuses on good program structure and on whether a program ‘works’. This leaves students with the tacit belief that security is less important than functionality [8].

These challenges can be overcome if there is an extensive effort by faculty members to integrate secure coding principles into undergraduate programming courses. An important factor in the drive to include comprehensive secure coding principles is to create a ‘security mindset’ amongst faculty members and students. Many experts consider this approach to be one of the most important strategies in making progress towards improving the implementation of secure coding education in undergraduate programming courses [22].

Thus we acknowledge the need for a structured method of integrating secure coding principles into undergraduate programming curricula similar to [16]. The ‘pervasive theme’ approach provides a structured method for achieving this. Pervasive themes are defined as topics that are addressed multiple times from different perspectives [16]. This means that the teaching of similar content within different undergraduate programming courses can occur. Teaching in this manner is necessary for a topic such as secure coding because students need to learn secure coding as part of all programming courses holistically and repetitively. The suggestion is that of considering the use of pervasive themes across undergraduate programming curricula.

A further approach to be considered is the ‘pillars-first’ approach which provides students with the core understanding of foundational software development concepts before integrating the pervasive themes. The main disadvantage of the pillars-first approach is that it tends to present each core concept in an isolated manner [16]. This problem can be addressed by a joint effort by faculty members by lecturing in a manner that enables students to understand how the various core concepts are connected.

In this paper we propose the application of the pillars-first approach as it allows fundamental concepts to be understood by students first, before various secure coding concepts are introduced across the several modules taught within an institution’s undergraduate programming curriculum. This approach also enables lecturers to gradually integrate secure coding into the curriculum without diminishing the focus on the core programming concepts [16]. The content for the students can then be taught by the lecturers at a rate they consider feasible for the given course. The importance of repeatedly addressing secure coding across several programming courses in computing curricula ensures that students understand that secure coding is an integral part of software development and not just an optional ‘nicety’.

5 A Phased Approach for Integrating Secure Coding Principles into Undergraduate Programming Curricula

The following sub-sections outline *three phases* that specify how secure coding principles can be integrated into undergraduate programming curricula. These phases are: *identification*, *buy-in*, and *implementation*.

5.1 Identification Phase

The identification phase (Fig. 1) is the first phase—motivated by the requirements of industry and its need for academia to teach undergraduate programming students secure coding. These requirements may change over time such that it is essential for universities to keep up-to-date with these changes. Similarly, the standards and best practices relating to secure coding and related vulnerabilities may also change. It is thus necessary that universities take cognisance of this when identifying which secure coding principles to integrate into their programming courses. Currently, the secure coding principles of [6, 18, 19, 23] are noteworthy. Furthermore, universities should refer to the relevant ACM curricular documents for Computer Science, Information Technology, Information Systems, etc. Figure 1 depicts the identification phase with these key elements, resulting in a set of relevant secure coding principles.

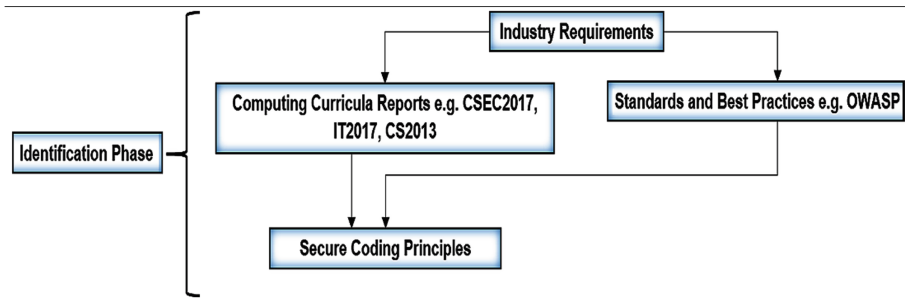


Fig. 1. Organisation of the identification phase

5.2 Buy-In Phase

The buy-in phase is the second phase, which requires lecturers to take the initiative in choosing the secure coding principles to be included into the ‘learning outcomes’ of their courses. Such an initiative would need a mixture of ‘top down’ and ‘peer-to-peer’ communications to satisfy both the head of department’s overall strategy as well as the teachings staff’s individual concerns. The initiative to integrate secure coding principles into the various programming courses must be supported and initiated by the faculty’s leadership. The buy-in of leadership is crucial for the success of such a long-lasting measurement. In [11], for example, we can find an organizational model for a university according to which a ‘director of school’ acts at the strategic level, a ‘head of department’ at the tactical level, and ‘lecturers’ at the operational level (as shown in Fig. 2).¹

¹ Less hierarchic models of academic organisation are known in the universities of other countries where the professors of a discipline constitute a ‘subject group’ without a formal director. Nonetheless, our curricular proposals can be implemented by such collaborative professoral ‘subject groups’ as well.



Fig. 2. Organisational model of [11]

Table 2. Example checklist for integrating secure coding principles

SECURE CODING PRINCIPLE	STUDY-YEAR
Input Validation	1, 2, 3
Authentication and Password Management	2, 3
Session Management	2, 3
Access Control	2, 3
Cryptographic Practices	3
Error Handling and Logging	1, 2, 3
Data Protection	2, 3
Database Security	3
File Management	2, 3

Furthermore, academic staff need a mechanism for determining which secure coding principles should be integrated into which programming courses and at which appropriate year of study. The observation of pervasive themes can guide a faculty in the understanding that each secure coding principle can be taught repeatedly across different course levels albeit with a different focus at each level. Table 2 shows an example of how an academic department can create a checklist for formally integrating secure coding principles into an undergraduate programming course in each year of study.

A secure coding checklist forms part of this phase due to the need for a formal structure for determining learning outcomes for each level of study in undergraduate programming curricula. Each department would need to customise such a checklist according to prior knowledge of its students as well as the structure of the curriculum in question. As an example, Fig. 4 shows a snippet of what can be taught to students for input validation [13]. The organisation of the buy-in phase is sketched in Fig. 3.

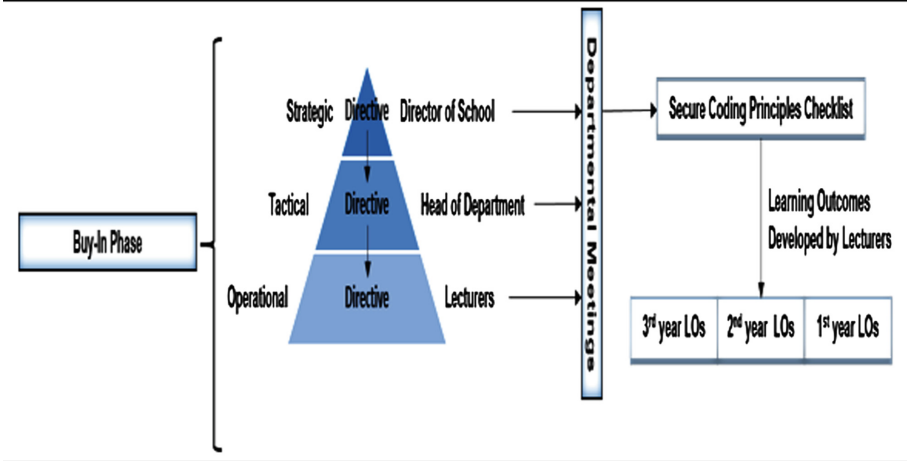


Fig. 3. Organisation of the buy-in phase

```
<form action="/action_page.php" method="post">
  <input type="text" name="fname" required>
  <input type="submit" value="Submit">
</form>
```

Fig. 4. Input validation example, from [13]

Figure 5 depicts the hierarchial learning model of Bloom’s taxonomy [17]. For the sake of simplicity we assume in this paper that the ‘remembering’ level is most appropriate for study-year 1, the ‘understanding’ level most appropriate for study-year 2, and the ‘applying’ level most appropriate for study-year 3 (although in practice all levels of learning can occur to some lesser or greater extent in each study-year). Already in the buy-in phase, this hierarchy of difficulties ought to be remembered when determining the appropriate learning outcomes for each course at each level. Details follow in the subsequent phase.

5.3 Implementation Phase

The revised version of Bloom’s Taxonomy [17] defines a multi-level model of learning, arranged according to six cognitive levels of complexity of classifying thinking. For the sake of simplicity, we refer in this paper only to the bottom three levels to illustrate how secure coding principles can be aligned to each of the three years. “When each topic is presented with a Bloom-level of mastery, the instructor is better informed as to what level of mastery is expected; thus he will be able to determine the required time and necessary instruction to help the student achieve the proper knowledge level for each topic” [21]. The three bottom levels (Fig. 5) are defined as follows [17]:

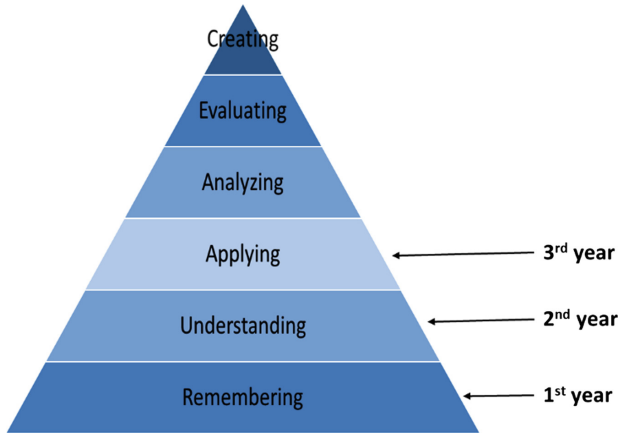


Fig. 5. Levels of difficulty, from [17], to be taken into account

Remembering refers to retrieving, recognising, and recalling relevant knowledge from long term memory.

Understanding refers to constructing meaning from oral, written and graphic messages through interpreting, exemplifying, classifying, summarising, inferring, comparing, and explaining.

Applying refers to carrying out or using a procedure through executing or implementing.

The bottom three levels of Bloom’s taxonomy can assist lecturers with devising learning outcomes that can be relatively understandable for undergraduate programming students. However, complexity of learning secure coding principles might perhaps become too difficult for students within the scope of a three-year undergraduate degree—for comparison see [14]—when the topic itself demands higher levels of mastery in the upper half of the pyramid. These levels could be addressed in fourth-year and postgraduate studies. Figure 6 sketches the integration of the lower levels of Bloom’s taxonomy into the planning considerations of the implementation phase.

Table 3 provides the relationship between the bottom three levels of Bloom’s taxonomy and the possible learning outcomes that can be implemented in each undergraduate year of study. Thereby, Table 3 is based on only one example—input validation—to illustrate how the learning outcomes for an identified secure coding principle can be determined by using appropriate levels of Bloom’s taxonomy. Since the complexity of secure coding principles varies at each level, the assessment at each level of teaching must also differ. Although this paper does not capture all details of all secure coding principles, Bloom’s taxonomy is well applicable throughout the implementation of the new curriculum.

The integration of secure coding principles in our ‘pillars-first’ approach allows students to learn secure coding principles in classroom settings and be assessed through periodic assignments and tests. However, the practicality of

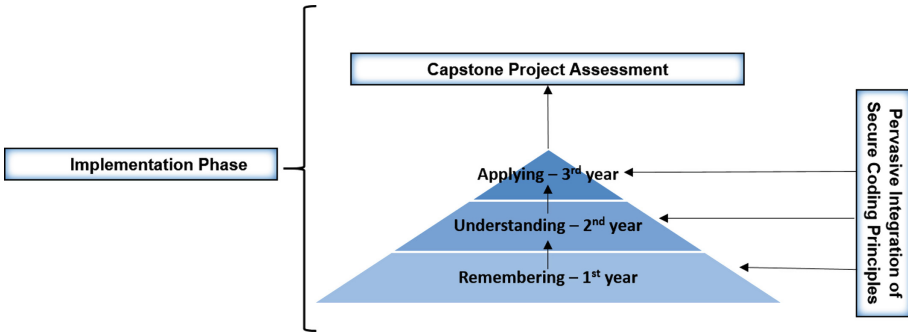


Fig. 6. Considerations for the implementation phase

Table 3. Example of ‘input validation’ mapped to learning outcomes

LEVEL	LEARNING OUTCOME FOR SECURE CODING PRINCIPLE
1st Year (remember)	<ol style="list-style-type: none"> 1. State why <i>input validation</i> is important 2. Define and provide examples of <i>input validation</i>
2nd Year (understand)	<ol style="list-style-type: none"> 1. Determine the types of <i>input validation</i> output in code segments 2. Explain how <i>input</i> is <i>validated</i> in various C# code segments
3rd Year (apply)	<ol style="list-style-type: none"> 1. Write C# code to <i>validate input</i> 2. Modify C# code to suit <i>input validation</i> principles

secure coding requires that a student’s learning progress be also assessed outside the formal classroom setting. According to [19], the IT curriculum must provide students with a ‘capstone’ experience that gives them a chance to apply their skills and knowledge to solve a challenging problem. The capstone project in undergraduate programming courses thus provides an opportunity for students to practically apply secure coding principles they have been taught. Therefore we include the capstone project into our curricular considerations, too.

Figure 7 shows the proposed framework with all three phases. The figure shows how the three phases are linked to each other for the sake of integrating secure coding principles into undergraduate programming curricula. The secure coding principles from the identification phase are approved and agreed upon by various computing faculty members in the buy-in phase. The lecturers of the various programming courses are responsible for determining the learning outcomes related to each secure coding principle according to the level of study at which they teach. These learning outcomes are then used as a basis for teaching and assessment during the implementation phase, whereby the capstone project in the 3rd year is the final assessment of students’ secure coding abilities.

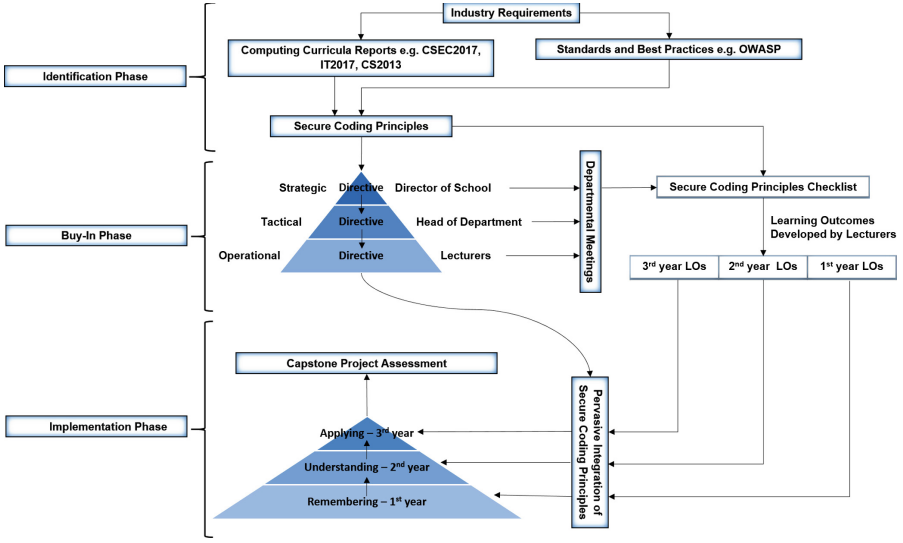


Fig. 7. Framework for integrating secure coding principles

6 Conclusion and Outlook

Evidence from the literature indicates that secure coding in undergraduate programming curricula is still not extensively taught. The literature emphasises the need for computing faculties in universities to consider integrating secure coding into their undergraduate programming courses to ensure that, upon graduation, these students can meet the software security needs of the IT industry. The various ACM Computing Curricula documents [6, 16, 19, 23] emphasise the need for secure software development, too.

The problem addressed by this paper is the general lack of formal inclusion of secure coding into undergraduate programming curricula. The proposed solution to this problem is the development of a framework to assist computing faculty members in this regard. The proposed framework incorporates a three-phased approach including: an identification phase, a buy-in phase and an implementation phase.

In the identification phase of our framework the ACM computing curricula recommendations can be used to justify the need for teaching secure coding principles in undergraduate programming courses, since the ACM considers the changing needs of the IT industry. This phase specifically assists with the identification of current standards and best practices related to secure coding by referring to the work of organisations such as OWASP.

The buy-in phase clarifies how multiple computing faculty stakeholders must be consulted to ensure the acceptance of the innovation at strategic, tactical and operational levels in the faculty. It includes the use of checklists to ensure

that the identified secure coding principles are addressed in several programming courses at various levels of study.

The implementation phase enforces the pervasive integration of secure coding principles into undergraduate programming curricula. Some of these principles are described in our foregoing paper [5], to which this paper can be read as a ‘continuation’. The use of Bloom’s taxonomy in the implementation phase ensures that well-defined learning outcomes are set appropriately for the relevant years of study. In addition, we suggest that a capstone project be used to ensure that undergraduate programming students learn to practically apply secure coding principles about which they must also be systematically assessed.

The purpose of our framework is to guide the integration of secure coding principles into undergraduate programming curricula by illustrating the key components and role players to be considered. The current limitation of our work is that it has not yet been validated by any actual implementation. Future work shall be dedicated to implementing our framework at various South African universities.

References

1. ACM: Key Education Activities. <https://www.acm.org/education/about-education>
2. Agama, E., Chi, H.: A framework for teaching secure coding practices to STEM students with mobile devices. In: Proceedings of the ACM Southeast Regional Conference, pp. 1–4 (2014)
3. Aratyn, T., Kazerooni, S.: Secure Web Application Framework Manifesto (2010)
4. Aziz, N.A., Shamsuddin, S.N.Z., Hassan, N.A.: Inculcating secure coding for beginners. In: Proceedings of the ICIC International Conference on Informatics and Computing, pp. 164–168 (2016)
5. Bangani, S., Futcher, L., van Niekerk, J.: An approach to teaching secure programming in the .NET environment. In: Tait, B., et al. (eds.) SACLA 2019. CCIS, vol. 1136, pp. 35–49 (2020)
6. Burley, D., Bishop, M., Buck, S., Ekstrom, J., Futcher, L., Gibson, D.: Cybersecurity Curricula. Technical report (2017)
7. Choudhury, A.J., Kumar, P., Sain, M., Lim, H., Hoon, J.L.: A strong user authentication framework for cloud computing. In: Proceedings of the APSCC IEEE Asia-Pacific Services Computing Conference, pp. 110–115 (2011)
8. Dark, M.J., Lauren, S., Ngambeki, I., Bishop, M.: Effect of the secure programming clinic on learners’ secure programming practices (2016)
9. Dark, M.J., Ngambeki, I., Bishop, M., Belcher, S.: Teach the hands, train the mind — a secure programming clinic. In: Proceedings of the 19th Colloquium for Information Systems Security Education (2015)
10. Duong, T., Rizzo, J.: Cryptography in the web: the case of cryptographic design flaws in ASP.NET. In: Proceedings of the IEEE Symposium on Security and Privacy, pp. 481–489 (2011)
11. Gomana, L.G.: Towards a framework for the integration of information security into undergraduate computing curricula. Masters dissertation, Nelson Mandela Metropolitan Univ. (2017)

12. Ingham, K.L.: Implementing a successful secure coding continuing education curriculum for industry: challenges and successful strategies. In: Proceedings of Software Engineering Education and Training Workshops, pp. 1–11 (2006)
13. Javascript. <https://www.w3schools.com/js/validation.asp>
14. Khomokhoana, P.J., Nel, L.: Decoding source code comprehension: bottlenecks experienced by senior computer science students. In: Tait, B., et al. (eds.) SACLA 2019. CCIS, vol. 1136, pp. 17–32 (2020)
15. Kindy, D.A., Pathan, A.S.K.: A survey on SQL injection: vulnerabilities, attacks, and prevention techniques. In: Proceedings of the ISCE International Symposium on Consumer Electronics, pp. 468–471 (2011)
16. Lunt, B., et al.: Information technology: curriculum guidelines for undergraduate degree programs in information technology. ACM/IEEE Joint Technical report (2008)
17. Orey, M., Forehand, M.: Emerging perspectives on learning, teaching, and technology (2011)
18. OWASP: Secure coding practices quick reference guide. Technical report (2010)
19. Sabin, M., et al.: Information technology curricula. Technical report. ACM (2017)
20. Sadeghi, A.R., Wachsmann, C., Waidner, M.: Security and privacy challenges in industrial Internet of Things. In: Proceedings of the DAC Design Automation Conference, pp. 1–6 (2015)
21. Starr, C., Manaris, B., Stalvey, R.: Bloom’s taxonomy revisited: specifying assessable learning objectives in computer science. In: Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education, p. 22 (2008)
22. Taylor, B., Bishop, M., Hawthorne, E., Nance, K.: Teaching secure coding: the myths and the realities. In: Proceedings of the 44th ACM Technical Symposium on Computer Science Education, pp. 281–282 (2013)
23. The joint task force on computing curricula: curriculum guidelines for undergraduate programs in computer science. ACM Technical report (2013)
24. Visaggio, C., Blasio, L.C.: Session management vulnerabilities in today’s web. *IEEE Secur. Priv.* **8**(5), 48–56 (2010)
25. Whitney, M., Richter, H.L., Chu, B., Zhu, J.: Embedding secure coding instruction into the IDE: a field study in an advanced CS course. In: Proceedings of the 46th ACM Technical Symposium on Computer Science Education, SIGCSE 2015 pp. 60–65 (2015)



Developing a Digital Forensics Curriculum: Exploring Trends from 2007 to 2017

Roshan Harneker¹(✉)  and Adrie Stander² 

¹ Department of Information Systems, University of Cape Town,
Cape Town, South Africa

`roshan.harneker@uct.ac.za`

² Digital Forensics and E-Discovery Research Unit,
Leiden University of Applied Sciences, Leiden, The Netherlands

`stander.a@hsleiden.nl`

Abstract. The young science of digital forensics has made great strides in the last decade, but so, too, has cyber crime. The growing complexity of cyber crime has necessitated that traditional forensics methods be updated to accommodate new technologies, and that further research is carried out to keep up with the rate of technological innovation. The main purpose of this paper is to determine how academic teaching and research can support the needs of the industry in investigating cyber crime. Current digital forensics curricula in higher education are discussed, followed by an analysis of academic research trends for this discipline for the years 2007 to 2017. We conclude by highlighting trends for which more research is required and which could possibly contribute towards shaping future teaching and learning of digital forensics in higher education.

Keywords: Digital forensics · Trends · Curriculum

1 Introduction

The proliferation and accessibility of the Internet have indelibly changed our lives in many positive ways. The Internet has, amongst others, improved cross-border collaboration, enabled almost instantaneous communication, and brought vast amounts of information to our fingertips at the click of a button. However, the Internet's accessibility has also led to a cyber crime explosion, whereby it is estimated (for example) that South Africa alone loses several billions of Rand annually to cybercrime.

Digital forensics, regarded as a relatively young science [12], is an emerging area found under the broader umbrella of *computer security* that is mainly concerned with the discovery and preservation of evidence in a digital format for proof of criminal behaviour and ultimately prosecution of criminal activity [1].

For a new discipline there is a need for the creation of a digital forensics *taxonomy* to guide the academic teaching to ensure that industry expectations

and academic offerings are aligned. As technological change progresses at a rapid speed, such a digital forensics taxonomy should be updated regularly to ensure that academia keeps up with industry's needs. There is also a requirement to support overburdened law enforcement agencies that need to keep up with ever-changing technological trends and the ways these are used to commit cyber crime.

In this paper we outline our research objectives and research limitations before moving to a definition of 'digital forensics', Higher Education Institutional (HEI) curricula, and current challenges. This is followed by a description of our methodology, a trend analysis, our results, and our summary and conclusions.

Our work aims to determine digital forensics trends covering the years 2007 to 2017 by investigating digital forensics (DF) trends published in academic resources. It also aims to highlight the current state of digital forensics research and, where possible, the needs of an industry that academic research should shift its focus to.

By highlighting certain trends, explaining their significance, and making recommendations on trends requiring more research, our analysis shall also assist with emphasising specific knowledge areas and with differentiating them from the more general knowledge areas.

There is always a possibility that the data may not match the research questions, or that it will contain particular gaps. Another limitation likely to be experienced is that *many papers conflate information security with digital forensics*. Many papers, therefore, had to be scrutinized for proper digital forensics contents before we were able to decide whether to include them into our research data set.

2 Related Work

2.1 Digital Forensics

Reith (et al.) distinguish between computer forensics and digital forensics by asserting that the former pertains specifically to methods used to find digital evidence on computers, while the latter uses scientifically verifiable methods to preserve, collect, validate, identify, analyze, interpret documents, and present evidence in digital form to be able to reconstruct incidents deemed to be of a criminal nature [13].

2.2 Higher Education Institutional Curricula and Challenges

Lang (et al.) highlighted some of the obstacles encountered when attempting to formulate an academic curriculum for digital forensics [9]. They found a lack of a standard curriculum and HEI-appropriate textbooks. Forensics training and education has a significant reliance on an instructor's or lecturer's personal experience. Moreover, the lack of a globally accepted curricular model can also contribute to institutions not adopting a forensics programme due to uncertainty

and impedance to curriculum development. Lang (et al.) also emphasise that digital forensics as a discipline straddles the areas of computer science and law [9]. Knowledge from both these fields is therefore a requirement—however: students studying digital forensics are highly unlikely to be studying both disciplines. This results in difficulties in deciding which prerequisites from each field students should have to meet, and which concepts from each field should be included in the curriculum.

Gottschalk (et al.) highlighted further difficulties pertaining to digital forensics training which is reliant on an instructor’s personal experience [7]. This can turn out to be problematic due to a shortage of qualified digital forensics practitioners. Hence it can be difficult to find qualified academics to provide training in an HEI setting.

To date no generally accepted model for a digital forensics curriculum exists, although there are some curriculum standard *proposals*. In this context is interesting to note that the fast pace at which the discipline is growing, and the generally slow pace at which academic learning material is created and altered to keep up with current digital forensics trends, did not feature as a ‘challenge’ to the curriculum designers.

For further recent work on these (and similar) topics see also [2, 10, 14, 15].

3 Method

Our descriptive review is meant to reveal patterns in the analyzed literature on the basis of quantifiable data such as publication time, the research methods underlying those papers, as well as their research results. Our method is *bibliometric*—i.e.: mostly using searching, filtering, and classification. We conducted a thorough and extensive literature review for relevant papers that pertain to the research area. Each paper is then treated as a single data record. This is followed by identifying trends and patterns. The result is claimed to be an accurate ‘snapshot’ of the current situation.

It is not practical to explore the totality of the field using interviews with academic or industry experts in the Digital Forensics field, or using questionnaires. Therefore we opted to use a descriptive literature review to determine what *new topics* emerged in the field and to get an indication of the relative importance of their *focus areas*.

Academic data was collected with help of Mendeley, a desktop- and web-based program that is used to manage and share research papers. Mendeley’s use also encourages collaboration and the discovery of research data online. We opted for Mendeley’s search function to avoid any bias which may arise from using specific databases such as ScienceDirect, EBSCOHost, IEEE Xplore, or the ACM DL. Search results were then further narrowed to the years from 2007 to 2017. Only the search term ‘digital forensics’ was entered. As the papers available via Mendeley are crowd-sourced and show the number of times an article has been read, we believe that Mendeley provides a reliable source of well-read peer-reviewed quality papers that have already been selected by a large pool of

independent researchers. We added a further delimiter to the retrieved papers by only choosing ones read by at least 5 readers.

Once enough peer-reviewed references were retrieved, the actual papers were downloaded via their hosting databases, websites or other sources. We collected 2200 articles and citations which were scanned manually for relevance by first reading abstracts and keywords. In cases where title, abstract and keywords did not provide enough information, the entire document was read to determine its relevance for our study.

During a second round of data analysis, we scanned abstracts (and read full texts if required) to exclude papers that did not have Digital Forensics as their central theme but merely mentioned it along with other interest areas. We also found papers which merely gave Digital Forensics a rather un-specific general coverage. Thus we filtered the papers that could not be placed into any relevant specific category. The papers which contained Digital-Forensics-related themes but could not be placed into a specific category were then placed into a 'general' category.

A third round of analysis involved reading all the remaining papers, and then applying *open coding* to ascertain and label variables in the form of categories, concepts and properties, as well as their interrelationships. The codes were generated from keywords as well as from analyzing the abstracts and the content of each paper.

4 Results

Our data analysis of 2200 papers, according to the method described above, revealed ≈ 50 trends. These are summarised in the following table—and further discussed thereafter—whereby the trend labeled '*General*' (RANK 6) consisted (as mentioned above) of a range of papers that either did not fit any of the specific categories, or where the topic of research was fairly broad. 'General' is thus not regarded as a trend in itself, though the papers listed under 'General' are still DF-related. Thus we consider our analysis as having revealed 49 *specific* trends, (not 50).

RANK	DIGITAL FORENSICS TREND 2007–2017	#PAPERS	%PERCENT
1	DF Process	173	8.33
2	Cloud Forensics	148	7.13
3	Image Forensics	141	6.79
4	DF Tools	128	6.16
5	Mobile Forensics	117	5.63
6	<i>General</i>	82	3.95
7	Digital Evidence	74	3.56

(continued)

(continued)

RANK	DIGITAL FORENSICS TREND 2007–2017	#PAPERS	%PERCENT
8	Network Forensics	73	3.51
9	Legal	70	3.37
10	Digital Forensics Framework	66	3.18
11	Education	62	2.99
12	Cyber crime	61	2.94
13	Digital Forensics Challenges	53	2.55
14	Hardware Forensics	52	2.50
15	Operating Systems Forensics	51	2.46
16	Information Security	51	2.46
17	Memory Forensics	49	2.36
18	Multimedia Forensics	48	2.31
19	Digital Forensics Standards	39	1.88
20	Malware Forensics	38	1.83
21	Virtualization	33	1.59
22	Internet Forensics	31	1.49
23	Live Forensics	29	1.40
24	Anti-Forensics	28	1.35
25	Digital Forensics Readiness	28	1.35
26	Email Forensics	26	1.25
27	Steganography	26	1.25
28	OSINT Forensics	25	1.20
29	Cryptography	24	1.16
30	IoT Forensics	23	1.11
31	Software Forensics	21	1.01
32	Database Forensics	20	0.96
33	Digital Forensics Trends	20	0.96
34	Big Data	16	0.77
35	Biometrics	13	0.63
36	Digital Records Forensics	13	0.63
37	Console Forensics	12	0.58
38	Drone Forensics	11	0.53
39	GPS Forensics	11	0.53
40	Incident Response	11	0.53
41	Peer 2 Peer Forensics	11	0.53
42	Digital Forensics Research	10	0.48

(continued)

(continued)

RANK	DIGITAL FORENSICS TREND 2007–2017	#PAPERS	%PERCENT
43	eDiscovery	10	0.48
44	Visualisation	10	0.48
45	Digital Forensics Analysis	8	0.39
46	SCADA	8	0.39
47	Bitcoin	7	0.34
48	Encryption	6	0.29
49	FaaS	5	0.24
50	Machine Learning	5	0.24
	TOTAL	2077	100

4.1 The Most Important Trends

Trends, generally, demonstrate a pattern of change in output, state or process, or the generalized inclination of a series of data points and the directions in which they shift over a period. Looking for consequential, relevant and significant trends is an important and prevalent undertaking in scientific work, whereby the statistical noteworthiness of a linear trend plotted against a time series is regularly used to classify and quantify the ‘usefulness’ of a trend observed [3].

Trend 1: Digital Forensics (DF) Process. The *Digital Forensics Process* is recognised as a valid scientific and forensic method used to conduct Digital Forensics investigations. It is defined as the steps to be taken from the time an alert of an incident is received to the time of the formal reporting of the analytic findings. These processes are mostly conducted on computing devices, including mobile ones, and the steps mentioned above follow the route of acquiring an image, analysing the image, and providing a written report of the investigation’s findings [4].¹ In our case, the trend encompassed a range of processes and/or procedures that were proposed for use for investigations that do not necessarily fit the mould of a traditional DF-related case. Notably, the year 2013 had a ‘dent’ in the number of papers about processes. No papers in the data sample showed process-related papers. This does of course not mean that no papers were written that year—only that none were found with Mendeley as auxiliary tool. The analysed papers discussed digital forensics case reconstruction, chains of custody processes, text string searching, how to conduct investigations, processes to use for embedded systems, hashing, data classification, insider threats, as well as processes pertaining to log gathering and analysis. Some of the more interesting papers discussed the use of digital forensics for medical cases and pattern matching by means of artificial intelligence. A variety of process methodologies and models were also described together with practical use cases.

¹ An internationally *standardized* definition of the term ‘*Digital Forensic Process*’ can be found in **ISO 27043**.

Much research has gone into the development of processes to follow when conducting DF-related investigations. As technology changes, this topic will no doubt continue to attract much research interest. 173 peer-reviewed papers of this topic were analysed, i.e.: 8.33% of our entire data sample.

Trend 2: Cloud Forensics. Cloud computing delivers services via shared pools of configurable computing system resources and is an ever-present transformative technology that is well known for its flexibility, scalability, elasticity, and consistency of service. It has changed the way in which data is created, stored, managed, used, shared and secured [1].

Zawoud and Hasan explain that cloud forensics is often considered as part of network forensics since cloud computing services require substantial network access, and network forensics investigations are conducted on private and public networks and the IP space [16]. Cloud forensics also includes the investigation of operating system processes, file systems, registry entries, and caches of the participating machines. Different forensics steps must be followed depending on which implementation model of cloud computing is involved. For example, collecting evidence for SAAS relies solely on the cloud service provider to obtain and send application logs, whereas with IAAS the data owner can obtain a virtual machine image directly from customers using the cloud service. This allows a forensics practitioner to examine and analyse the images.

Although cloud forensics is commonly thought of as a subset of network forensics, the research on this topic was significant enough to be considered a trend on its own. Cloud forensics came to rank 2 in our table with 148 peer-reviewed research papers, i.e.: 7.1% of the entire data sample. With the move away from private physical infrastructure towards cost-saving cloud solutions, this topic will continue to garner interest and research. Interestingly, however, research seems to wane after 2016 according to our data sample. We believe that this could be due to Digital Forensics being lumped more and more into information security research.

Since many cloud solutions are cross-jurisdictional there are also legal implications that affect where organisations and individuals store their data. Clouding is also a move away from traditional computing upon which traditional Digital Forensics methods are based. Clouding has become a ubiquitous part of life given that cloud features are built into most current smartphone and tablet mobile devices—making it both a consumer and enterprise product. Cloud investigations can stymie those who are used to the concept of taking custody of a hard drive to forensically image and analyse it, as the hard drive is not physically present on the computing device used to access the cloud service which is often accessed via a web client. The existence and use of cloud computing and its associated services such as IAAS, SAAS, and PAAS meant that new, forensically sound methods needed to be developed to acquire and analyse cloud-based data. Here we must bear in mind the different ways in which the cloud will affect the ability of a forensic practitioner to obtain the data required in a forensically sound manner.

Trend 3: Image Forensics. Image forensics refers to the processes followed to analyse and investigate digital photographic images. This should not be confused with forensic photography which refers to pictures taken at and of crime scenes for a court of law. Kim (et al.) explain 5 classification types of image forensics techniques [8]: pixel-based, format-based, camera-based, physically-based, and geometry-based. Farid describes these techniques in more detail [6]. Pixel-based techniques identify statistical deviations introduced at the pixel level and can analyse interconnections that occur because of image tampering. Format-based techniques analyse statistical associations that arise from a specific lossy compression scheme. Camera-based techniques highlight artefacts introduced by the camera’s lens, sensor or onboard processing chip. Physically-based techniques model and highlight irregularities in the interaction between the camera, physical objects, and light. Lastly, geometry-based techniques measure objects being photographed and their position in relation to the camera photographing them.

The technology of today caters for almost imperceptible changes to be made to digital media that would not have been possible as recently as 20 years ago. The plethora of papers noted for image forensics—141 in total—made this trend the third most important one in our data sample assessed, (6.79% of all papers analysed). Most of those papers focused on forgery and image manipulation. Several papers described methodologies and algorithms to detect anomalies and variances from original images. The number of papers per year that contributed to this trend reached a maximum in the year 2009 and a minimum in 2017.

Trend 4: Digital Forensics Tools. This trend refers to the array of tools available for imaging, indexing and analysing digital forensics images and data artefacts. These tools are commonly used for cases that may be tried in a court of law. They must thus withstand legal scrutiny and satisfy legal requirements. 21 out of the 128 analyzed articles delved into the use of open source tools and their associated merits. Interest in this topic peaked in 2013; the 2017 yielded no such papers with our method of search. Related topics included the use of tools for automation of manual tasks, challenges associated with the use of DF tools, and using tools for investigation standardisation (amongst others).

Trend 5: Mobile Forensics. This trend covers digital forensics conducted on mobile devices including cell phones and tablets. According to [11], the influx of smartphone devices on the consumer market resulted in a burgeoning demand for digital forensics that could not be met by traditional forensics investigative techniques. There were 117 papers (5.63%) covering this trend which reached its peak in the year 2013 when smartphone usage became more ubiquitous. The papers assessed discussed operating systems forensics for Android, iOS, and Windows smartphones, legal issues pertaining to the use of cell phone data, the development of frameworks specifically for mobile device forensics, application and software forensics for mobile devices, and data recovery. Marturana (et al.) further observe that law enforcement officials are more than likely to encounter criminals with at least a smartphone in their possession than a larger computing device such as a laptop or desktop [11]. This trend in the ‘top 5’ also relates to the evolution of investigations from ‘live forensics’ which consists of examining

mobile content via the screen in a decidedly non-forensic manner. It, therefore, became important to create and streamline image acquisition, indexing, and analysis techniques that could be conducted with forensics in mind, and therefore also withstand legal scrutiny in a court of law.

Quasi-‘Trend’ 6: General. This quasi-‘trend’ is actually only a label to list all assessed papers which remained uncategorized either due to the broadness of their topics or due to too few other papers with similar content. These papers consisted of a wide range of topics covering the detection of hoaxes, fraud, and deception based on online writing style, how forensics is being shaped, and many others. In total, 86 papers fell into this category.

Trend 7: Digital Evidence. Casey defines ‘digital evidence’ as data in a binary form that is transmitted via or stored on a computing device that either supports or refutes a hypothesis held about how an offense has taken place or that speaks to certain aspects of the offense such as intention or alibi [4]. Data comprising digital evidence consists of either text, images, audio or video or a combination of these elements. Digital evidence has, in the past, been submitted to courts of law in the form of emails, word processor documents, GPS coordinates, digital photographs, computer printouts, backups, and computer memory to name a few. This topic consisted of 75 papers (3.56% of the total data sample). 74 of those were peer-reviewed; 1 came from a popular media sources. They discussed automated production of digital evidence, a network-based architecture proposal for the storage of digital evidence, guidelines for seizing, imaging and analysing digital evidence, the need for standardising digital evidence, how to manage digital evidence, challenges facing digital evidence, court judges’ awareness of digital evidence, how to assess whether digital evidence is forensically sound, and digital evidence for mobile devices. There was an almost consistent interest in this theme between 2009 and 2015, with far fewer papers published from 2016 onwards.

Trend 8: Network Forensics. Network forensics, according to [1], forms part of network security which addresses the requirement for dedicated investigative competencies to be able to investigate the origin and traversing of malicious network traffic—which constitutes security attacks—by dealing with the acquisition, recording, and analysis of network-related events for law enforcement purposes. 73 papers were analysed for this trend with discussions of intrusion investigations, analysis of VoIP traffic, proposals of network forensics frameworks, IP traceback models, analysis of wireless network traffic, connection chain analysis, network security, locational wireless and social media surveillance, wireless security vulnerabilities, evidential discovery of networked smart devices, organisational network forensics readiness, network analysis of the ToR network, network forensics education, network forensics challenges, and analysis of honeypot traffic, to name a few. Most of those papers were written in 2010 and then again in 2014. Fewer papers in this category appeared from 2016 onwards—perhaps due to the tie-in between this topic and network security which is a subset of

information and cyber security. This topic constituted 3.51% of the total number of papers analysed.

Trend 9: Legal Matters. This trend refers to the legal aspects of digital forensics. As the 9th trend in our list it consists of 73 papers, 70 of which are peer-reviewed; (2 were duplicates and 1 came from a popular media source). This trend comprised 3.37% of our total data sample. These papers discussed legal issues affecting digital forensics tools, forensics and the legal system, the validation of digital evidence for legal argument, bridging differences in digital forensics for law enforcement and national security, forensic analysis of a false digital alibi, investigating and prosecuting cyber crime, digital forensics and legal systems across different countries, legal and technical issues affecting digital forensics, and digital forensics testimony in courts of law. Interest in this research topic peaked in 2008 and then again in 2011, but declined from 2016 onwards. This is decline of interest is peculiar, as digital forensics is a process that exists primarily for courts of law.

Trend 10: Digital Forensics Frameworks. This very important field of research addresses frameworks for digital forensics, of which many have been proposed since this science first emerged. At present, there is no de-facto framework that acts as a one-size-fits-all. Since digital evidence can be found on almost any computing device, several frameworks exist to cater for the different hardware and software technologies. What remains constant is that the methods used to extract and analyse data for a digital forensics investigation must withstand legal scrutiny. This trend accounts for 68 papers (3.18% of the entire sample) of which 66 were peer-reviewed, (1 was a duplicate and 1 was from a popular media source). The papers discussed digital forensics investigative frameworks, forensics frameworks for web-related services, triage frameworks for digital forensics, open source frameworks for digital forensics, frameworks for analysing internet-related traffic, frameworks aimed at enhancing timeline analysis during a forensic investigation, disk monitoring and analysis frameworks, frameworks for hybrid evidence investigation, and a case-based reasoning framework aimed at improving the trustworthiness of forensic investigations.

Trend 11: Education. This trend comprised 62 papers (2.99% percent of the total data sample) with 2010 as the year in which most of its papers were published. The discussion in these papers focussed on various education programmes and curricula in use in countries around the world, on incorporating digital forensics understanding into law school programmes, creation of practical lab exercises for students studying forensics, case studies in teaching forensics, defining agendas for forensics education, assessment strategies for forensics training, as well as teaching forensics in different operating system environments.

YEAR	MOST-RESEARCHED TREND
2007	Digital forensic process
2008	Digital forensic process
2009	Image forensics
2010	Image forensics
2011	Image forensics
2012	Cloud forensics
2013	Digital forensic process
2014	Cloud forensics
2015	Cloud forensics
2016	Cloud forensics
2017	Cloud forensics

4.2 Data Analysis of Papers by Years

From a starting point in 2007 with 120 papers, the field showed consistent growth until 2016 with 279 papers, at which point it began to taper off. This may be attributed to the increase in academic research focused on information and cyber security. Facets of forensics have been absorbed into information security, such as incident response and general forensic and cyber security readiness, which follow similar methods to achieve their respective aims. However, it is recommended that future research be conducted to fully explore and compare the number of papers submitted relating to digital forensics and information security respectively. Another possible cause could be the stagnation of developments in the field at that stage. This is likely to change with many recent developments that incorporate machine learning and artificial intelligence.

By year, the following most researched topics trends were observed; The small table of above shows that the top 3 trends for this period are digital forensic processes, image forensics, and cloud forensics.

5 Conclusions and Outlook to Future Work

We suggest that still more research is required to determine the digital forensics trends that are important for curriculum development for HEI. For this purpose we analysed a significant sample of publications that dealt with digital forensics trends. Practitioners' and academic interest in digital forensics continues, following the trends in cyber crime. While we cannot claim this paper to be exhaustive, it provides insights into digital forensics trends previously researched. This overview could be valuable to researchers and/or experts who are looking for further direction w.r.t. where to focus their teaching, learning, and publication efforts. This paper shall, in particular, contribute towards the design of curricula, as it points out areas of interest that might otherwise be overlooked, such as cloud forensics, digital image forensics, and investigation frameworks.

We pointed to a range of topics that have seen significant research already and are thus important for inclusion into forthcoming digital forensics-related HEI curricula. Based on our findings we can emphasise cloud forensics, mobile forensics, digital forensics processes, image forensics, and digital forensics tools for this purpose. Cloud and mobile forensics, as discussed above, tend to move away from traditional forensics techniques, processes, and methodologies. They are complemented by forensics processes and forensics tools which have had to evolve to accommodate this move away from traditional forensics methods. Image forensics remains relevant due to several factors: cameras being incorporated into mobile and smartphone devices, the rise of social media, the use of photography, and the increase in the use of technology to commit cyber crime by altering digital images.

With our data sample it was not yet possible to fully determine the scope of forthcoming digital forensics curricula in HEIs. However, our data sample was able to comprehensively determine where academia had concentrated its digital forensics research efforts. 49 distinct trends were identified. Future research should also address the trends highlighted via popular media, as the corporate world tends to advance and adopt technology at a faster rate than HEIs do.

There is also a need to determine why the number of papers published on digital forensics seems to be declining despite the ever-growing urgency for organizations to be able to conduct digital forensic investigations caused by the sharp increase in cyber crime. There would be value in determining whether other disciplines, e.g. information and cyber security, are incorporating aspects of digital forensics into their research agendas. Lastly, another area of forensics in need of active research is that of standardisation, not only w.r.t. investigative methodologies (see ISO 27043), but also and especially HEI curricula, as this fledgling discipline continues to grow and evolve in complexity as a result of the fast rate of technological change and the globally sharp rise in cyber crime.

References

1. Almulhem, A.: Network forensics: notions and challenges. In: Proceedings of IEEE ISSPIT International Symposium on Signal Processing and Information Technology, pp. 463–466 (2009)
2. Bagby, J.W.: The cyber forensic war room: an immersion into IT aspects of public policy. In: Carroll, J.M. (ed.) *Innovative Practices in Teaching Information Sciences and Technology: Experience Reports and Reflections*, pp. 117–132. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-03656-4_11
3. Bryhn, A.C., Dimberg, P.H.: An operational definition of a statistically meaningful trend. *PLoS ONE* **6**(4), e19241 (2011)
4. Casey, E.: *Digital Evidence and Computer Crime: Forensic Science, Computers, and the Internet*. Academic Press, Orlando (2011)
5. Endicott-Popovsky, B., Frinke, D.: Embedding forensic capabilities into networks: addressing inefficiencies in digital forensics investigations. In: Proceedings of IEEE Workshop on Information Assurance, pp. 133–139 (2006)
6. Farid, H.: Image forgery detection. *IEEE Sign. Proc. Mag.* **26**(2), 16–25 (2009)

7. Gottschalk, L., Liu, J., Dathan, B., Fitzgerald, S., Stein, M.: Computer forensics programs in higher education: a preliminary study. *ACM SIGCSE Bull.* **37**(1), 147–151 (2005)
8. Kim, H.J., Lim, S., Kim, B., Jung, E.S.: A new approach to photography forensics using 3D analysis for correcting perception errors: a case study. In: *Proceedings of ACM Workshop on Surreal Media and Virtual Cloning*, pp. 27–30 (2010)
9. Lang, A., Bashir, M., Campbell, R., de Stefano, L.: Developing a new digital forensics curriculum. *Digit. Investig.* **11**, s76–s84 (2014)
10. Leung, W.S.: Cheap latex, high-end thrills: a fantasy exercise in search and seizure. In: Liebenberg, J., Gruner, S. (eds.) *SACLA 2017. CCIS*, vol. 730, pp. 265–277. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-69670-6_19
11. Marturana, F., Me, G., Berte, R., Tacconi, S.: A quantitative approach to triaging in mobile forensics. In: *Proceedings of the IEEE TrustCom 10th International Conference on Trust, Security and Privacy in Computing and Communications*, pp. 582–588 (2011)
12. Olivier, M., Gruner, S.: On the scientific maturity of digital forensics research. In: Peterson, G., Sheno, S. (eds.) *DigitalForensics 2013. IAICT*, vol. 410, pp. 33–49. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-41148-9_3
13. Reith, M., Carr, C., Gunsch, G.: An examination of digital forensic models. *Int. J. Digit. Evid.* **1**(3), 1–12 (2002)
14. Stenvert, M., Brown, I.: Qualifications and skill levels of digital forensics practitioners in South Africa: an exploratory study. In: Kabanda, S., Suleman, H., Gruner, S. (eds.) *SACLA 2018. CCIS*, vol. 963, pp. 345–361. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-05813-5_23
15. Wu, D., Fulmer, J., Johnson, S.: Teaching information security with virtual laboratories. In: Carroll, J.M. (ed.) *Innovative Practices in Teaching Information Sciences and Technology: Experience Reports and Reflections*, pp. 179–192. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-03656-4_16
16. Zawoad, S., Hasan, R.: Cloud forensics: a meta-study of challenges, approaches, and open problems. Technical report, [arXiv:1302.6312](https://arxiv.org/abs/1302.6312) (2013)

Software Engineering Education



Hackathons as a Formal Teaching Approach in Information Systems Capstone Courses

Walter F. Uys^{1,2} 

¹ SDL Research Focus Area, North-West University, Mahikeng, South Africa
walter.uys@nwu.ac.za

² CITANDA, University of Cape Town, Cape Town, South Africa

Abstract. Hackathons are ‘hacking marathons’ in which participants collaboratively and rapidly prototype new applications over a 24–48 h period. The potential of hackathons as a strategy for stimulating interest in the CS fields is well known. Hackathons share many similarities with capstone courses, however their application as a formal teaching approach in the CS/IS curriculum is less prevalent. This paper describes the introduction of a curricular hackathon in a 3rd-year IS capstone course at a South African university. An exploratory case study was conducted to evaluate feedback from the participants and organizers. In the process, the students completed seven new applications which they had conceptualized during the course. They also learned something about new technologies and programming interfaces as well as they exhibited growth in personal and inter-personal competencies. Seven fundamental differences between curricular and traditional hackathons are highlighted. Suggestions for integrating hackathons into undergraduate CS/IS capstone courses are provided together with possible areas for further research.

Keywords: Information systems education · Capstone courses · Software application development · Hackathons · Project-based learning

1 Introduction

Application development is one of the fastest growing high-paying careers in the USA [6]. Because software development environments are free and easily accessible, there is a perception that anyone can become a successful developer without any formal education or training [6]. Contrasted against this growth in demand for application developers is the *concern about the value of a university degree versus practical experience* [12, 16, 31, 48, 50].¹ Thus there remain concerns w.r.t. the *workplace-readiness* of graduates as well as a general lack of soft-skills that are needed in the workplace [3, 21, 40].

¹ See also <https://www.diskonto.net/2019/02/18/skills-vs-degrees-debate/>.

In higher education institutions (HEI) there is a fair understanding of this need, and the *capstone course* is one of the strategies used to fill this gap [50]. Capstone courses provide students with the opportunity to integrate theoretical and practical aspects of the curriculum in such a way as to develop a real-world project that has some benefit to society [27]. There are different models of capstone courses ranging from limited support and no classes (the traditional model) to clearly defined deliverables, extensive tutor/lecturer support and scheduled classes and/or meetings [2]. Most courses find a balance between these two models. Because capstone courses are mainly student-driven through ‘learning by doing’, the role of the lecturer and lectures are less clear. The role of the lecturer is to transition students from academic/theoretical studies towards real-world professional practice. Some guidance in this process is useful, however indications are that fostering a real-world environment that encourages active learning strategies has greater benefit than the minimally guided approach. There are many active teaching/learning strategies that can be implemented as an instructional design in CS/IS to narrow the theory/practice gap and develop some of the ‘soft-skills’ that are globally needed in the software- and IT-industry: see [32] for comparison. The primary strategy adopted in capstone courses is referred to as *project-based learning* (ProjBL) [22, 26] which is not to be confused with problem-based learning (PBL) [60]. Other strategies that can be used are experiential learning [14], work-integrated learning [59], case-based learning [56], game-based learning [23, 38] as well as virtual learning [36]; for a broader overview see [13]. Though these strategies have some commonalities, project-based learning emphasizes an educational strategy aimed at solving real-world project-based problems [22].

An under-represented approach for implementing ProjBL in CS/IS capstone courses is the hackathon [15]. Whilst colleges and universities are frequently the preferred setting for hackathons [33, 44, 57], they are mainly used as an informal approach to expose the youth to CS/IS and leverage their creativity [29, 44, 52]. Hackathons share many key characteristics with capstone courses [41], yet their use in the software engineering (SE) and computer science curriculum is not widely reported [18, 41, 42]. This means that educators have minimal guidance/tips or techniques for introducing hackathons in their courses.

This paper responds to this desideratum by showing how hackathons can be used as a formal educational approach for implementing ProjBL in the CS/IS curriculum—specifically in our *3rd-year Information Systems Development* course for the *Bachelor of Commerce* (BCom) degree. The secondary research question is about the differences between traditional versus curricular hackathons. We describe our experiences of introducing a curricular hackathon in our course at a South African HEI. We do so by outlining the central concepts of hackathons, recapitulating related work, outlining our research method, and presenting our findings and recommendations for implementing hackathons in the future under-graduate (UG) IS curriculum.

2 Central Concepts and Related Work

Hackathons originated in 1999 from the voluntary efforts of programmers in order to develop/advance a free/open-source operating system called OpenBSD [45]. Although the concept of a hackathon is in need of a more precise definition [24], we adopted the following

Working Definition: *Hackathons are events where computer programmers and others involved in software development, including graphic designers, interface designers and project managers, collaborate intensively on software projects in a short period of time, typically 24–36 h [34].*

There are many different kinds of hackathons—each with a unique approach. Some are referred to as data-dives, code-fests, code-jams, hack-days, sprints, edit-a-thons [35], data-thons [1, 4], code-camps [42] or game-labs [23]. Over time these events have increasingly become sponsored by corporations such as Facebook [9], F-Secure [24] or KPMG, as well as by governmental agencies such as Gov-Hack [43], CivHack [17] or NDPHack.² These sponsorships have transitioned hackathons from their philanthropic roots to become more competitive with teams keeping their innovations closely guarded until they are presented for adjudication [45]. Motivation for participating in hackathons is mainly for financial gain, personal development, having fun, or “*the opportunity to meet new people while learning and experimenting with technologies*” [24].

Hackathons promote innovation in product and application development, new uses for existing products or apps or new solutions for government, business or education [25, 46, 51, 52, 54]. Ideas or innovations may be bottom-up or top-down [24], i.e.: originating either from the developers or from senior management, thereby fostering an entrepreneurial approach. They typically take place over extended and focused periods of between 24 and 72 h in dedicated venues [34]. Participants remain primarily at the venues with limited breakaways for ablutions or eating and optional sleeping during such events [46] although there are reports of virtual participation [33]. Catering such as food, energy drinks and coffee is normally provided [24] and infrastructure such as computers and data projectors may be available, although participants are normally encouraged to provide their own laptops or hardware devices [10].

The targeted participants for hackathons are mainly software developers and technical personnel, although teams may be comprised of programmers, analysts, designers, subject-matter experts, managers and community representatives [46]. Team sizes can vary between three and five people, with anything from five to fifty teams competing at a particular event [57]. Hackathons can also specifically target under-represented groups such as women and historically disadvantaged individuals (HDIs) [7, 24, 53]. There is also general consensus that (good) programmers are a scarce resource at hackathons [45]. Furthermore, it is acknowledged that financial and material support by leadership is important to hosting such events [46].

² See <http://www.ndp2030hackathon.gov.za/ndp-2030>.

Hackathons can either be closed or open events. Closed events are internal to organizations [24]. Open events are organized as public or civic events that are open to everybody [17]. Open events are broadly publicized and attract large corporate sponsorship to encourage attendance and participation [24]. Open events mostly focus on a specific topic or theme such as health and fitness [24], health-care [52,55], Internet of things (IoT) or wearable devices [10], whereas closed events might be geared around a new product feature or innovations for a specific company such as Facebook [9,24].

Hackathons are able to provide “*new and exciting opportunities for education and research*” [23] as well as to develop project management and communication skills in addition to creativity and innovation amongst participants [18]. They are, however, known to restrict sound software-architectural approaches to development and provide only limited testing and quality-assurance opportunities [44]. It is also questionable how much new (programming) skills can be acquired during such short events. The perception remains that participants rely on familiar skills and techniques [45] and have only “*limited time to interact with industry experts or learn valuable hard and soft skills relevant to SE*” [41].

Though there is a shortage of published reports of formal hackathons in the UG CS/IS curriculum [18,41,41], there are some related studies that can provide insights into how to introduce those into the curriculum. For example, [19] describes a method for implementing a Hackathon for UG course projects. That approach followed a software development methodology over a period of 24 h with 22 students divided into seven groups. The purpose of that hackathon was to develop an Internet-of-Things application using *Raspberry Pi* devices. The event was hosted outside the university according to a traditional hackathon format. The event was however bespoke and took students through the entire SDLC and was thus not integrated into a capstone course.

A follow-up paper by the same author [18] describes the experiences of developing student projects in a continuous 10 h ‘Hack Day’ on a Saturday at the end of the semester. Although a ‘Hack Day’ does not have the full time commitment of a 24–72 h Hackathon, it appears to offer similar benefits. Students were organized into groups of three. They were co-located to avoid groups working remotely as well as to fostering collaboration amongst students and allowing the instructor to provide support. Feedback was obtained from the 24 students in the course, whereby the results indicated that the students learned more during the experience than with traditional classes, and that the event fostered greater motivation and engagement amongst the students and closer relationships with the instructor. Further feedback indicated a high degree of task focus during the event as well as an increase in time management skills as a result of the time pressure during the event.

The principles of hackathons were also applied to address student learning outcomes in a first-year Engineering curriculum [44]. Even though this intervention was targeted at achieving course-level objectives, the approach followed included ‘design days’ which bear minimal resemblance with informal hackathons. The main outcome was new designs (not software applications).

The purpose of those ‘design days’ was to improve collaboration between students and staff, to expose students to engineering design concepts, to integrate knowledge from across the curriculum, and to stimulate creative thinking. The primary curriculum outcomes of the ‘design days’ were improved teamwork, better understanding of design, and greater student engagement. Non-curricular outcomes were a highly creative and fun/engaging event for the students as well as increased motivation to participate in such events.

Another study examined the efficacy of the hackathon approach to stimulate students’ enrollments and interest in CS [33]. Six hackathons were hosted at a university over a period of three years. Participation was voluntary after the students had been invited to participate in the event. The hackathon gave students an opportunity to learn and network with subject matter experts and to be part of larger project teams that were focused on rapidly developing socially relevant solutions. The primary outcomes raised students’ exposure to mentoring, work-integrated learning, and collaborative learning. Limited integration with the formal coursework was achieved due to the open nature of the hackathon. The curricular benefits were latent, with students reporting an improved social and practical understanding of CS concepts, further developing their interest and passion for participating in the field, as well as of changing their perceptions of CS by-and-large.

As it can be seen from the basic definition of hackathons and from how they have been used in higher education in the case studies recapitulated above, there remains the *question of how a hackathon can be better integrated into the curriculum*. This question is addressed in the subsequent sections.

3 Methods and Materials

For this paper an *exploratory case study* [61] was carried out. This method is suited for novel studies where the experiences of participants and the context of action is important [5]—in our case: normal classroom activities and student evaluations. In compliance with our university’s regulations, participants’ names and identities have been kept anonymous, and statements were attributed with three-letter acronyms (TLA) representing the corresponding persons. The participants completed an informed-consent form which outlined the purpose of our study, the use of the students’ answers, and the confidentiality of their information. The students were also told that their voluntary participation would have no bearing on their final marks (course results). Our case study was supported by first-person reports and reflections, additional reports by the hosting organization, as well as anonymous student feedback and evaluation. No personal interviews were conducted. The answers obtained were analyzed by means of *topical analysis* [47] which can reveal key topics or issues in a corpus of text, a discourse or some particular event. Topical analysis provides a method for comparing and analyzing similarities and differences between related topics, definitions, artifacts or concepts [20].

4 Case Study

There are sufficient commonalities between a traditional hackathon and our curriculum hackathon for us to refer to our object of study as ‘hackathon’. Some of the main similarities between the two are that our event was a focused event that occurred at a particular venue (a computer lab on campus), over a fixed period of 24h. Catering, coffee and drinks as well as PCs were provided and attendees had access to a kitchen and ablution facilities. Students and lecturers were encouraged to stay awake and present at the venue for the entire period although there were some exceptions.

The event was scheduled over a Friday/Saturday and timed closer to the end of the second semester of the South African academic year (19–20 October) so that there were no conflicts with other courses, assignments, tests or exams. After the hackathon, our students had 2–3 weeks to complete their documentations for examination purposes. Students were divided into seven teams of three students each. There were different roles in each team, such as project manager, analyst, developer.

4.1 Aims and Objectives

The emphasis at our hackathon was the *delivery of a working system*, whereby the documentation for the system was drafted afterwards for assessment purposes. Although there were no incentives offered at the hackathon, students were informed that the top three teams would be selected *to participate* at the above-mentioned SITA NDP2030 hackathon which itself carried a prize of 100'000 South African Rand (\approx US\$6'000) for the winning team.

4.2 Phases

The typical hackathon can be represented by means of the classical IS Input-Process-Output model [24]. The input phase is the pre-hackathon phase where ideation and team building take place. The process phase is the actual hackathon where intense ‘hacking’ occurs and results are demonstrated. The post-hackathon phase is where teams decide to continue with the idea, form new teams or grow the teams and adopt new technologies and develop plans for funding.

In our case the pre-hackathon phase took approximately 12 weeks which overlapped with the traditional capstone curriculum. During this time, the students formed their teams, conceptualized their ideas, developed a business case, designed their apps, started building them and elicited requirements from other stakeholders. One of the teams was also responsible for planning the event and had to facilitate the event t-shirts, catering and permission for the event. The week before the event, the preparations began in earnest and all students were involved in final preparations for the event. On the final days before the event, drinks and meals were purchased and the catering orders confirmed.

The hackathon event itself (process phase) started at 18h00 (planned was 17h00) on the Friday, and, after initial presentations and motivation by the

organizing team, the students had dinner. After dinner the students ‘hacked’ for 4h and presented their progress to the entire forum at midnight. After the presentations the students had snacks and then continued to hack till 05h00 when they presented their results again to the forum.

Departing from the traditional hackathon approach we had some physical exercise in the morning of the second day whereby some sports games were played by the students to get energetic again after a long night of intense coding. Thereafter, the students started hacking again from 08h00 till 12h00. At noon the students had lunch and resumed hacked again 13h00–17h00; then they submitted their final projects. According to one participant (TLM),

“the results that we reached at the hackathon were amazing: we managed to get most features of the application working during those 24h”.

At the conclusion of the formal hackathon activities, students, lecturers and facilitators were treated to a barbecue. Thereafter, our students were so exhausted that they were glad to ‘pack up’ and go home.

During the post-hackathon phase (output phase), the students made the final changes to their apps in order to capture screenshots for their project report and also to prepare for their final assessment presentations and demonstrations. They were also required to write individual reports to explain their individual contributions to the their groups. For this report they had been advised already at the beginning of the semester to keep a record of their personal tasks and activities. The final demonstrations and presentations were held four weeks later. This gave the students the opportunity to explain their solutions to invited representatives from government, industry and the university.

4.3 Projects

During the first semester (February to June) of our academic year, our students were required to develop a project plan that included the business case, user requirements, project scope and costs as well as high-level designs and GANTT charts. In the second semester the students started implementing these system development projects. These ideas were initially conceptualized by the students and implemented through various iterations and interactions with the lecturers and stakeholders. Only during the hackathon the students completed and presented their final systems; (see Table 1).

4.4 Results

The course’s lecturers found that the event gave the students the opportunity to focus solely on the completion of their projects without other distractions. The event also imposed personal challenges to the students, such as to stay awake for its entire duration and to work under pressure. During the hackathon, students learned how to work with new technologies, tools and software development platforms. For example, they learned how to develop in Java on Android Studio;

Table 1. App development: teams and systems

TEAM/APP	SPECIFICATION
Residence Control System	Monitoring and managing visitors access to student residences to avoid illegal squatting
e-License App	A mobile app to register and represent ‘virtual’ driver licenses. It must allow road officers to validate physical and virtual licenses as well as to check for outstanding fines or license expiry. It has a front-end sub-app and back-end sub-app
<i>Soapy Shine</i> Car Wash	A car wash loyalty app that allows members to check a shop’s availability (or current queue length), to book and be notified at any of the participating shops
Billboards Innovation	Facilitates the remote management and hosting of advertisements on electronic billboards
Stay Residence Booking System	A variation of ‘AirBnB’ for student accommodation: intended mainly for finding and booking of ‘approved’ university ‘digs’
Virtual Housing Project	A VR system to convert 2D plans into 3D virtual walkthroughs for visualizing the architectural features of real estate properties
Clinic Appointments	A queuing system for public hospitals and clinics, similar to those found in banks or other mobile service providers. It must be able to distinguish emergency cases from dispensary patients

they learned database development as well as the use of technologies such as XAMMP, PHP, or UMAJIN. They were exposed to cross-platform development architectures and the use of APIs (such as Google’s authentication features and maps in Android Studio) as well as to interfacing with bulk SMS providers which had not been part of their under-graduate coursework so far. Consequently,

*“I got to understand more about cross-platform development. I did more research on it **during the hackathon** as we were busy with the coding and development. I also learned more about APIs as we had to do research on how we’re going to integrate them into our application to work better with other existing applications (i.e. Google Maps)”* (KJT).

Surprising were the unintended ‘soft skills’ the students developed during the project; see [40] for comparison. The students learned much about teamwork, project management (the first semester complement of this module), time management under pressure, punctuality, responsibility, creativity, bug-fixing as well as presentations an audience. Although these skills are not considered to be the main outcomes of an IS course [28], they are recognized as critical for the successful integration of graduates into the workplace [11, 48]. Thus the hackathon

can be regarded as a suitable ‘active learning’ method for developing these ‘soft skills’ which are hard to teach in the lecture hall.

The students also provided some insightful comments as to the efficacy of the approach through entries in the university-administered anonymous *Student Experience Survey* that was completed online at the end of the semester; (see Table 2). In particular:

“*The project on its own requires that each individual has to play a role in making progress. There is no time for dependency. Teamwork pays off, but, most importantly, the ability to communicate with other people is very essential when you are working on something so volatile*” (ZAN).

Table 2. Learning experiences and suggested improvements

What did you like about the teaching and learning experiences of this module?	Stud.
<i>To communicate and be able to work in a group, to participate in class. This class prepares us for the life outside university</i>	(ST1)
<i>The experiences were great; it taught us about our pre-professional lives</i>	(ST2)
<i>It also taught me to grow as an adult and be more responsible</i>	(ST3)
<i>The learning styles used</i>	(ST4)
<i>Creativity</i>	(ST5)
<i>It is like doing real events that affect my life in a very tangible way</i>	(ST6)
What suggestions do you have to improve the teaching and learning experiences of this module?	
<i>This module needs _[more] time to be able to complete the project as there is so much to do</i>	(ST7)
<i>Introduce the work-integrated learning strategy in other modules, _[too]</i>	(ST8)
<i>Every group must be assigned a supervisor</i>	(ST9)
<i>The use of a study guide</i>	(ST10)
<i>More practicals, field trips and recruiting of sponsors for practicals (like Microsoft, hackathon- and other IT-related companies)</i>	(ST11)
<i>The class experience was very new; I don't think there is much more that I can add</i>	(ST12)
<i>It's perfect!</i>	(ST13)

Without a suitable control group it is difficult to assess the effect or outcomes that the hackathon had on the quality of the students’ solutions, i.e.: what they learned during the process as compared to a traditional approach. Even the students’ final marks would not provide a fair representation of learning, as students and project topics in other years would differ individually. Assessing software development progress in industry is an ongoing challenge [39],

though the usual metrics (such as lines of code, function points or completed features) could also be used to assess the value that the hackathon had on the students' projects progress. Ultimately, the measure of success for such an approach would be to 'track' these students into the industry and see how they are faring there in practice.

5 Discussion

Some similarities and differences between the curricular and the traditional hackathon are explored next, as per our secondary research question.

5.1 Curricular Hackathons

Scope and Purpose. The scope and purpose of the curricular hackathon is much narrower than the traditional hackathon, which typically takes an idea from concept to prototype stage during the event. The objective of the curricular hackathon is to provide the students with a focused 24 h period in which to complete the projects that they had been working on over the course of the semester. The purpose for introducing the hackathon was in response to the problems that students were experiencing in completing their projects during the semester due to competing demands from other courses.

Conceptualization of Projects. Unlike in traditional hackathons, the ideas for the projects originated mainly from the students themselves. This was due to limited participation from outside stakeholders. In [33], by contrast, project ideas originated from schools, non-profit organizations, expert and, in some instances, from computer students. After coming up with their ideas, however, our students were encouraged to engage with other lecturers and industry stakeholders in the field for which the solutions were intended. This meant that the projects or ideas were not necessarily aligned with 'national priorities'. It is suggested that in curricular hackathons, ideas for projects such as those from the list of national priorities are given to students to choose from.

Time Frame. As the hackathon was held at the end of the capstone course, students had the entire year to conceptualize and plan their projects. The students were required at the beginning of the year to produce ideas for innovative apps that address some organizational, societal or academic needs. In the first semester the students developed the business case, user requirements, system requirements, prototypes and project plans. In the second semester they did systems analysis and design, system architecture, use-cases, and user interfaces; then they and started building the system.

Closed Event. Another difference between the two is that curricular hackathons are closed events as opposed to open events for traditional hackathons. Our event was restricted to 3rd-year IS students who were enrolled in the course. The reason for excluding other students were that this group of students had

been working on their projects from the beginning of the year. Inviting other participants at such a late stage would have detracted from the educational focus described above, and would also have disadvantaged those participants who had not already been working on a project during the course of the year.

Incentives. Also, unlike in a traditional hackathon, students earned marks for completing particular aspects of their solution throughout the year. There were key points where students needed to interact with stakeholders from industry in order to develop their ideas and designs as well as to present them to lecturers in the faculty. Marks were allocated for documentation, apps and presentations during the semester according to formative assessments. The final assessment was a presentation of students' working projects to industry stakeholders, lecturers from the department, and the external examiners. Students were also evaluated on the project documentation as well as the software code that was submitted at the end of the hackathon. All the material that was developed by the students during the course was uploaded to the institutional E-Learning system.

Compulsory. In contrast to the free open culture of traditional hackathons,³ curricular hackathons restrict the voluntary nature of traditional hackathons, yet still allow for the philanthropic [45] and socially relevant [33] ideals. Firstly, participation in the hackathon was compulsory. Secondly, students were required to develop socially relevant solutions, although this might not always be feasible, especially if organizational pressures or corporate funding prevails. Thirdly, it is conjectured that students were not necessarily motivated by the social cause of their solution, but by obtaining marks, and thus may have complied with the design brief merely to pass the course. They still had, however, a large degree of freedom in choosing which topic they wanted to focus on as well as the technologies or design scheme they wished to apply.

Intellectual Property. Questions were also raised by the students as to the ownership of the intellectual property that was developed during the hackathon. It was suggested that the same regulations that pertain to academic research be applied to hackathons. In the end, however, it appears as if few students or groups intended to incubate or continue with the projects which they developed during the hackathon. The sustainability of such projects should be designed into the activities in order to move away from disposable assignments to renewable projects [58].

5.2 Teaching Approach

One of the clear advantages of integrating the hackathon into the traditional capstone course [49] is that it puts students into a high-pressure, team-based learning environment where they need to perform much like in industry. By contrast, the traditional capstone course imposes no such demands, resulting in students remaining undecided on particular system or technology decisions that

³ For comparison see <http://www.gnu.org/philosophy/free-sw.html>.

need to be made, and rushing their projects at the end of the semester. Some other challenges and opportunities with this approach are discussed below.

All-nighter. Some concerns have been raised as to the effects of sleep deprivation amongst the students. Research shows that students are accustomed to ‘all-nighters’ due to academic and social pressures and that acute sleep deprivation can have a physical but not necessarily a cognitive impact on healthy university students [37]. In order to ameliorate this effect (based on the organizers past experience of 72 h hackathons) an exercise session was held in the morning to boost the participants vitality. These concerns can also be addressed by changing the format to a ‘code camp’ [42] which is a coding event that is hosted over the period of a week during normal class times. The test week would be an ideal time for such a ‘code camp’ and would provide greater opportunities for the students to interact with industry experts and to develop the soft skills that are expected of software engineers.

Facilitation. Another limitation of the hackathon approach is that it requires additional management, teaching facilitation, time and resources from the lecturer and department that are not necessarily catered for by traditional curriculum teaching activities. The process can be facilitated by external parties that ease the transition of hosting a hackathon; however this will incur additional costs. In our case, we used the services of a professional organization (PRO) to run the event as well as the residence catering services to provide the food. PRO was responsible for advertising the event, managing its schedule as well as providing transport and accommodation for the mentors that were brought in from other companies and regions.

Mentoring. In addition, the curricular hackathon emphasizes an apprenticeship model where students are guided by experienced mentors from industry, lecturers and senior students. The mentors were responsible for motivating the students at the start of the event, and for providing feedback and advice during presentations throughout the night as well as technical advice. Because the event was held on campus after academic hours, we needed to obtain permission from the campus security services, the director of student life, the students’ representative council, and the manager of the soccer institute where the end function was held. This was all arranged by one of the groups of students with the guidance of the lecturer.

Accelerating Projects. Additionally, we found that the hackathon was effective in accelerating the completion of student projects, especially at the end of the semester when they were pressured by other courses to prepare for exams and final reports. Finally, we learned that a curricular hackathon can be a fun event that stimulates students’ interests in the discipline as well as ‘awe’ amongst non-participants.

Closed Event. One concern raised by students from other years and other departments was why they could not participate in the curricular hackathon. This was explained to them due to the closed nature as part of the IS capstone course. Our suggestions are now to host two hackathon events during the year:

The first event at the start of the year should be open to all students in order to expose them to the concept and to allow the third-year students to conceptualize their projects. A second, closed event should be held towards the end of the semester and be restricted to the final-year IS students for them to complete their projects for marks.

Assessing Progress. It is difficult to assess the degree of software development progress that groups can achieve during hackathons. We suggest that source-code repositories such as *GitHub* be used to manage and monitor the software development progress throughout the semester and during the hackathon.

In summary, curricular hackathons are closed events that are directed at accelerating students' capstone projects in a focused 24 h session that is hosted on campus by experienced facilitators. Students are divided into teams of between 3 and 5 students at the start of the semester. The projects are conceptualized and developed during the semester and completed at the hackathon. Participation is compulsory. Projects (presentations) are assessed for marks during the hackathon and at the final project-day. These events require other expertise and resources to facilitate than traditional teaching and/or capstone projects. Last but not least the integration of academic (assessment) requirements into a traditional hackathon creates a number of additional challenges for the facilitators.

6 Conclusion

Hackathons provide a unique blend of active learning approaches [8, 10, 22, 23, 25, 30, 34, 44] in focused 24–72 h events. Although they are widely hosted at colleges and universities [33, 44, 57], their role as a formal teaching approach in the CS/IS curriculum was hitherto not well understood.

This paper describes the three phases [24] of a curricular hackathon conducted in a *3rd-year undergraduate capstone course for information systems design and development*. During a 12 week pre-hackathon phase the teams were formed, ideas conceptualized, projects planned and development commenced (much like a traditional capstone course). During the 24 h hackathon event the students completed their projects through a process of mentorship and regular feedback. Functional support for the students was provided by means of catering and other facilities. In a post-hackathon phase of 4 weeks, the students finalized their projects, completed their documentation, and presented their software apps to industrial and academic stakeholders.

Our students found that the hackathon was a valuable activity that prepared them for the demands of the industrial work environment. They learned more about working with new technologies and tools, doing cross-platform development, using Google APIs and public SMS services. More importantly they developed a number of important 'soft skills' [40] during the process. They learned much about teamwork, project management, time management under pressure, punctuality, responsibility, creativity, bug-fixing and presentation of achievements.

Thus a curricular hackathon is a viable approach to facilitating workplace skills in the CS/IS curriculum that complements traditional capstone courses. This paper provides some guidelines on how curricular hackathons can be implemented and highlights some of the most important differences and similarities in comparison to traditional hackathons. Some challenges and limitations of this approach are also outlined.

Future work should evaluate the efficacy of ‘code camps’ as opposed to curricular hackathons. Improved means of assessment are also called for in evaluating curricular hackathons. Further research may also look at what it means to ‘hack’, and how these approaches foster workplace skills amongst new generations of ICT students.

Acknowledgments. The research described in this paper was done as part of the author’s participation in our regional *Institutional Teaching Excellence Awards (ITEA)*. Thanks to the senior staff of our faculty who supported this project both financially and administratively; the directors of student life and campus security; the facilitation company who assisted us in planning and hosting the event; the management team from the student group who did additional work in order to ensure the event’s success; as well as the other lecturers and students who participated in the event. Thanks to the external examiner for suggesting such an educational intervention as well as for attending the final presentations and providing guidance during the process. Thanks for the financial contribution made by the schools’ directors towards hosting the hackathon. Last but not least thanks the anonymous reviewers for their critical feedback as well to the SACLA’2019 conference’s audience for some insightful remarks following the presentation of this work.

References

1. Aboab, J., et al.: A ‘Datathon’ model to support cross-disciplinary collaboration. *Sci. Transl. Med.* **8**(333), 1–5 (2016)
2. Adams, L., Daniels, M., Goold, A., Hazzan, O., Lynch, K., Newman, I.: Challenges in teaching capstone courses. *ACM SIGCSE Bull.* **35**(3), 219 (2004)
3. Andrews, J., Higson, H.: Graduate employability, ‘Soft Skills’ versus ‘Hard’ business knowledge: a European study. *High. Educ. Eur.* **33**(4), 411–422 (2008)
4. Anslow, C., Brosz, J., Maurer, F., Boyes, M.: Datathons: an experience report of data hackathons for data science education. In: *Proceedings of SIGCSE 2016*, pp. 615–620 (2016)
5. Benbasat, I., Goldstein, D.K., Mead, M.: The case research strategy in studies of information systems. *MIS Q.* **11**(3), 369–386 (1987)
6. Bhardwaj, P.: This is the Fastest-Growing Six-Figure Job in America and it Doesn’t Require a Degree (2019). <http://money.com/money/5635712/this-is-the-fastest-growing-six-figure-job-in-america-and-it-doesnt-require-a-degree/>
7. Blanco, L.: BeSmart hackathon: HBCU students hacked into their futures in silicon valley. *Black Enterp.* **49**(2), 54–57 (2018)
8. Bowen, L.M.: The limits of hacking composition pedagogy. *Comput. Compos.* **43**, 1–14 (2017)
9. Burnham, K.: Inside Facebook’s Hackathons: 5 Tips for Hosting Your Own (2012). <http://www.pcadvisor.co.uk/news/internet/3377232/inside-facebooks-hackathons-5-tips-for-hosting-your-own/>


10. Byrne, J.R., O'Sullivan, K., Sullivan, K.: An IoT and wearable technology hackathon for promoting careers in computer science. *IEEE Trans. Educ.* **60**(1), 50–58 (2017)
11. Calitz, A.P., Greyling, J.H., Cullen, M.D.M.: South African industry ICT graduate skills requirements. In: *Proceedings of SACLA 2014 Annual Conference of the Southern African Computer Lecturers' Association*, pp. 135–145 (2014)
12. Carr, W.: Theories of theory and practice. *J. Philos. Educ.* **20**(2), 177–186 (1986)
13. Carroll, J.M. (ed.): *Innovative Practices in Teaching Information Sciences and Technology — Experience Reports and Reflections*. Springer, Cham (2014). <https://doi.org/10.1007/978-3-319-03656-4>
14. Clark, R.W., Threeton, M.D., Ewing, J.C.: The potential of experiential learning models and practices in career and technical education & career and technical teacher education. *J. Career Tech. Educ.* **25**(2), 46–62 (2010)
15. Duhring, J.: Project-based learning kickstart tips: hackathon pedagogies as educational technology. In: *Proceedings of NCIIA*, pp. 1–8 (2014)
16. Gallet, F.: The Education vs. Experience Debate. *Bona Magazine* (2015). <https://showme.co.za/lifestyle/the-education-vs-experience-debate/>
17. Gama, K.: Crowdsourced software development in civic apps – motivations of civic hackathons participants. In: *ICEIS*, vol. 2, pp. 550–555 (2017)
18. Gama, K.: Developing course projects in a hack day: an experience report. In: *Proceedings of ITiCSE 2019, Aberdeen*, pp. 388–394 (2019)
19. Gama, K., Alencar, B., Calegario, F., Neves, A., Alessio, P.: A hackathon methodology for undergraduate course projects. In: *Proceedings of FiE 2018 Conference* (2019)
20. deGaray, J.: Aristotelism of difference. *Found. Sci.* **13**(3/4), 229–237 (2008)
21. Hendarman, A.F., Tjakraatmadja, J.H.: Relationship among soft skills, hard skills, and innovativeness of knowledge workers in the knowledge economy era. *Procedia Soc. Behav. Sci.* **52**, 35–44 (2012)
22. Janse van Rensburg, J.T., Goede, R.: A reflective practice approach for supporting IT skills required by industry through project-based learning. In: Kabanda, S., Suleman, H., Gruner, S. (eds.) *SACLA 2018. CCIS*, vol. 963, pp. 253–266. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-05813-5_17
23. Jennett, C., Papadopoulou, S., Himmelstein, J., Vaugoux, A., Roger, V., Cox, A.L.: Case study 3: students' experiences of interdisciplinary learning while building scientific video games. *Int. J. Game-Based Learn.* **7**(3), 93–97 (2017)
24. Komssi, M., Pichlis, D., Raatikainen, M., Kindström, K., Järvinen, J.: What are Hackathons for? *IEEE Softw.* **32**(5), 60–67 (2015)
25. Lara, M., Lockwood, K.: Hackathons as community-based learning: a case study. *TechTrends* **60**(5), 486–495 (2016)
26. Laware, G.W., Walters, A.J.: Real-world problems bringing life to course content. In: *Proceedings of 5th Conference on Information Technology in Education*, pp. 6–12 (2004)
27. Leidig, P.M., Ferguson, R., Leidig, J.: The use of community-based non-profit organizations in information systems capstone projects. *ACM SIGCSE Bull.* **38**(3), 148 (2006)
28. Loveland, T.R.: Teaching personal skills in technology and engineering education: is it our job? *Tech. Eng. Teach.* **1**, 15–20 (2017)
29. Lyndon, M.P., et al.: Hacking Hackathons: preparing the next generation for the multidisciplinary world of healthcare technology. *Int. J. Med. Inform.* **112**, 1–5 (2018)

30. Maaravi, Y.: Running a research marathon. *Innov. Educ. Teach. Int.* **55**(2), 212–218 (2018)
31. Merriam, S.: Some thoughts on the relationship between theory and practice. In: Merriam, S.B. (ed.) *New Directions for Continuing Education*, pp. 87–91. Jossey-Bass, San Francisco (1982)
32. Motta, G., Wu, B. (eds.): *Software Engineering Education for a Global E-Service Economy – State of the Art, Trends and Developments*. Springer, Cham (2014). <https://doi.org/10.1007/978-3-319-04217-6>
33. Mtsweni, J., Hanifa, A.: Stimulating and maintaining students' interest in computer science using the hackathon model. *Indep. J. Teach. Learn* **10**(1), 85–97 (2015)
34. Nandi, A., Mandernach, M.: Hackathons as an informal learning platform. In: *Proceedings of 47th ACM SIGCSE*, pp. 346–351 (2016)
35. Nolte, A., Pe-Than, E.P.P., Herbsleb, J., Filippova, A., Bird, C., Scallen, S.: You hacked and now what? Exploring outcomes of a corporate hackathon. In: *Proceedings of ACM Conference on Human-Computer Interaction*, p. 129 (2018)
36. Padayachee, I., van der Merwe, A., Kotzé, P.: Virtual learning system usage in higher education - a study at two South African institutions. *South Afr. Comput. J.* **57**, 32–57 (2015)
37. Patrick, Y., et al.: Effects of sleep deprivation on cognitive and physical performance in university students. *Sleep Biol. Rhythm.* **15**(3), 217–225 (2017)
38. Perrotta, C., Featherstone, G., Aston, H., Houghton, E.: *Game-Based Learning: Latest Evidence and Future Directions*. NFER Research Programme: Innovation in Education, Slough (2013)
39. Pfleeger, S.L.: Software metrics: progress after 25 years? *IEEE Softw.* **25**(6), 32–34 (2008)
40. Pieterse, V., van Eekelen, M.: Which are harder? Soft skills or hard skills? In: Gruner, S. (ed.) *SACLA 2016*. CCIS, vol. 642, pp. 160–167. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-47680-3_15
41. Porras, J., et al.: Hackathons in software engineering education – lessons learned from a decade of events. In: *Proceedings of ICSE*, pp. 40–47 (2018)
42. Porras, J., Knutas, A., Ikonen, J., Happonen, A., Khakurel, J., Herala, A.: Code camps and hackathons in education – literature review and lessons learned. In: *Proceedings of 52nd Hawaii International Conference on Systems Science*, pp. 7750–7759 (2019)
43. Powell, D.: Want to run your first Internal Hackathon? Here are some Tips from KPMG (2017). <https://www.smartcompany.com.au/startupsmart/advice/want-run-first-internal-hackathon-tips-kpmg/>
44. Rennick, C., Hulls, C., Wright, D., Milne, A.J., Li, E., Bedi, S.: Engineering design days: engaging students with authentic problem-solving in an academic hackathon. In: *Proceedings ASEE Annual Conference*, Salt Lake City (2018)
45. Richterich, A.: Hacking events: project development practices and technology use at hackathons. *Converg. Int. J. Res. New Media Tech.* **25**, 1000–1026 (2017)
46. Rosell, B., Kumar, S., Shepherd, J.: Unleashing innovation through internal hackathons. In: *Proceedings of IEEE Innovation in Technical Conference*, Warwick (2014)
47. Ross, W.D. (ed.): *The Works of Aristotle*, vol. I. Clarendon Press, Oxford (1928)
48. Scholtz, B., Cilliers, C., Calitz, A.: Bridging the Enterprise Systems (ES) skills gap: the South African challenge. In: *Proceedings of SACLA 2014 Annual Conference of the Southern African Computer Lecturers' Association*, pp. 3–5 (2010)

49. Scott, E.: From requirements to code: issues and learning in IS students systems development projects. *J. Inf. Tech. Educ.* **7**, 1–13 (2008)
50. Scott, E., Alger, R., Pequeno, S., Sessions, N.: The skills gap observed between IS graduates and the systems development industry: a South African experience. In: *Proceedings of Information Science and IT Education Conference, Cork* (2002)
51. Senghore, F., Campos-Nanez, E., Fomin, P., Wasek, J.S.: Applying social network analysis to validate mass collaboration innovation drivers: an empirical study of NASA’s international space apps challenge. *J. Eng. Tech. Manag.* **37**, 21–31 (2015)
52. Silver, J.K., Binder, D.S., Zubcevik, N., Zafonte, R.D.: Healthcare hackathons provide educational and innovation opportunities: a case study and best practice recommendations. *J. Med. Syst.* **40**(7), 177 (2016)
53. Tobor, N.: The Largest All-Female Hackathon in South Africa (2017). <https://www.iafrikan.com/2017/08/12/girlcode-the-largest-all-female-hackathon-in-south-africa/>
54. Tsjardiwal, L.: Hackathons: an effective communication tool for innovation within an organization? Thesis, Erasmus Universiteit Rotterdam (2016). <https://pdfs.semanticscholar.org/360b/1ebef9c6c75aafa20b6b7831a3aeb82a55b3.pdf>
55. Wang, J.K., Pamnani, R.D., Capasso, R., Chang, R.T.: An extended hackathon model for collaborative education in medical innovation. *J. Med. Syst.* **42**(12), 239 (2018)
56. Ward, R.: Active, collaborative and case-based learning with computer-based case scenarios. *Comput. Educ.* **30**(1/2), 103–110 (1998)
57. Warner, J., Guo, P.J.: Hack.edu: examining how college hackathons are perceived by student attendees and non-attendees. In: *Proceedings of ICER 2017 ACM Conference on International Computing Education Research*, pp. 254–262 (2017)
58. Wiley, D.: What is Open Pedagogy? (2013). <http://opencontent.org/blog/archives/2975/>
59. Winberg, C., Garraway, J., Engel-Hills, P., Jacobs, C.: Work-Integrated Learning: Good Practice Guide #12 (2011). http://www.ru.ac.za/media/rhodesuniversity/content/communityengagement/documents/Higher_Education_Monitor_12.pdf
60. Yaqinuddin, A.: Problem-based learning as an instructional method. *J. Coll. Phys. Surg. Pak.* **23**(1), 83–85 (2013)
61. Yin, R.K.: *Case Study Research: Design and Methods*, vol. 5, 2nd edn. SAGE Publ., Thousand Oaks (1994)



Modernizing the Introduction to Software Engineering Course

Marko Schütz-Schmuck^(✉) 

Department of Mathematical Sciences, University of Puerto Rico at Mayagüez,
Mayagüez, Puerto Rico
marko.schutz@upr.edu

Abstract. We describe the modernization of an undergraduate introductory course in software engineering that started in 2017–2018 (semester 2) offered at the University of Puerto Rico. We present the institutional setting, our underlying philosophy, and resources considered. We aimed at complementing informal descriptions in any phase with formal ones. We describe the revised course, discuss evaluations of the modernized course as held in two subsequent semesters, and outline options for future improvement.

Keywords: Software engineering · Formal methods · Education

1 Introduction

What should be taught in an introductory software engineering (I2SE) course and how should it be taught? Or, even more generally: What is software engineering, and what is engineering? After two colleagues retired who had been teaching our I2SE course, we took the opportunity to revisit these questions and to review the I2SE course. No simple or final answer would be expected. Nonetheless, the questions lead to valuable insights and allow us to discover our tenets.

We accept that engineering combines scientific knowledge with creativity and imagination to design an artifact and to show of this design that the resulting artifact will have desired properties. To predict the properties engineering uses scientific results and mathematics. Each of the sub-disciplines of engineering uses the branches of mathematics most appropriate for the type of properties of interest in the sub-discipline. Many of the sub-disciplines use differential equations for modeling and simulation, also calculus and linear algebra are frequently used.

Branches of mathematics relevant in software engineering are logic and proofs, algebra, and discrete mathematics. Properties of software-intensive systems can then be expressed and justified using such mathematics.

We believe that descriptions, statements, etc., in an adequate formal language should *complement* (not replace) any informal descriptions of properties of a

system-to-be or of its environment. Or, in the words of Mills: “*Natural language is imprecise; formal language is inaccurate*” [16]. This affects all phases and/or iterations into which we might choose to subdivide the software engineering process. The use of suitable formal languages thus cuts across the topics of our course.

We also believe the little use of formal texts is an indication of the immaturity of software engineering. Considering its relative age, it is not surprising that this discipline is less mature than (e.g.) mechanical engineering. First uses of the term ‘software engineering’ date back to the mid to late 1960s, so software engineering is now in its 50s. Mechanical engineering, in comparison, can be said to have started with Newton’s laws of motion —first published 1687— which make it about 330 years of age. This time has allowed the body of skills and knowledge in mechanical engineering to mature and stabilize while at the same time allowing society to form reasonable expectations towards the discipline. By contrast, the body of skills and knowledge in software engineering still seems much less stable and as society experiments with this young discipline it shows lenience towards the discipline’s failures. From the discipline’s successes and failures reasonable expectations have to form, and to the extent to which society increases its reliance on the outcomes of software engineering, the discipline will have to mature. To this end it will increasingly develop and incorporate scientific (in this case mathematical) foundations in similar ways as what can be seen in the more mature engineering disciplines [2].

We do not believe an introduction of formal texts to be a panacea. Conversely, we do believe that most phases of software engineering benefit from complementing informal with formal descriptions, justifications, etc., and that the mere attempt to express more formally an understanding of the domain, of requirements, etc., improves such understanding.

Coming back to our initial question: What should be taught in an introductory software engineering course, and how should it be taught? As it can be expected, the answer was already largely constrained. The existing programs at our university which require the I2SE course with the courses that build on it, the existing programs’ accreditation, the way the courses leading up to I2SE are taught, and the professional opinions of colleagues constrained and guided the exploration of this question. We detail the context in which we operate and the constraints arising from it in Sect. 3. We considered related courses at other universities, a selection of textbooks, and reports on innovative ideas on introducing rigor and formal reasoning into computing curricula. Only one of the textbooks considered combined formal and informal language in the way we wanted, namely the triple-volume of [3–5]. More related work is presented in Sect. 2. None of the related courses at other institutions seemed directly usable as a model for our revised course: either for lack of consistently complementing the formal with the informal or for lack of openly accessible information about details of the course. However, from Bjørner’s textbook and his direct advice we revised the course: Sect. 4 presents details of the way the course was offered in 2017–2018 (semester 2) and in 2018–2019 (semester 1). Students did not receive

the revised course as positively as we had hoped. Section 5 describes feedback from students' blogs and surveys of the two distinct editions held up to the time of this writing, but low student response rates to our surveys make conclusions rather difficult. For the upcoming edition we plan to make improvements in the delivery of the course. Our lessons learned and plans for the future of the course are in Sect. 6.

2 Related Work

Much has been written on the topic of software engineering education [17]. With a young (and rapidly changing) discipline like software engineering it is not surprising that the (introductory) courses of it would change rapidly as well. On the other hand, most educators will aim to balance immediately actionable content and timeless principles (and everything in between).

This can be achieved by going from the very concrete to the more principled. Abelson and Greenspun take this approach: in [1] they describe the development over several years of MIT Course 6.916: Software Engineering of Innovative Internet Applications. At the time of their writing this had developed into a survey course in which students completed 4–5 projects instead of the one semester-long project often found in similar courses. The topics were targeted very specifically to the students' projects and included user registration and management, discussion systems, voice interfaces, protocols (e.g. SOAP, WSDL, UDDI). Presumably, students are thus led from the concrete to the abstract and will learn about (e.g.) concurrency issues. Much of the audience of course 6.916 had *already completed* a software engineering lab at MIT. Their course was comparatively small (≈ 30 students) and they found student progress presentations in lecture-time greatly useful. They also had alumni willing to act as coaches to the student teams. The two variants of the course both have a common teaching goal: students' ability to *"take vague and ambitious specifications and turn them into a system design that can be built and launched within a few months"*. They conclude that their course supports this goal by having students build several applications in one semester, by their use of alumni as software engineering coaches, by the availability of a collaborative student workspace, by real clients, and by *"an emphasis on oral and written presentations"*.

In [25], Vallino and Basham describe the history of their introductory software engineering course of over 20 years during which the course was revised at least 8 times. Over time they used opposing approaches on some characteristics of the course: e.g.: 'heavy on process' vs. 'minimal process', Java vs. C++ vs. Python, or desktop applications vs. web applications. Their course had 450 students per year in up to 13 sections per semester. *Neither faculty nor students felt "like we had gotten it right"*. Two premises were identified as the cause: *"it needed to provide a broad overview of the software engineering discipline, and it would use one of the classic software engineering textbooks that covers all of those areas"*. A key insight here is that dropping some topics that are typically included in introductory courses does not harm: in the former version

of the course these topics were “*not being covered at a level that imparted that full understanding to begin with, and the software engineering students would see the full breadth and depth of those topics later in their program*”. One filter applied to the topics excluded all those on which students were merely required to remember knowledge. Their remaining design topics were: domain analysis, OO design, appreciation for software architecture, web architecture and development, domain-driven design, state-based behavior, unit testing, sequence diagrams, code metrics, and appreciation for usability. The authors decided not to use an encyclopedic textbook, and instead supplied a collection of resources including (e.g.) Youtube videos. For the course material that they developed themselves they followed a process very much like the process that the students were expected to follow on their course project. Based on their surveys the authors conclude that there remain areas for improvement, but that the revision has improved student perception of the course in most aspects.

We very much like the idea of following a similar process for the course material development as the one the students will follow on their projects. Aside from the benefits the authors mention, changing our view from ‘instructors developing course materials’ to ‘instructors developing a teaching and learning process’ makes the task amenable to the well-established process improvement practices.

These two publications are almost 20 years apart. Both agree that students should produce in the introductory software engineering course working implementations. Consequently, they lean towards the software design and implementation end of the development process and are less concerned with (e.g.) domain description or rigorous reasoning about properties of resulting artifacts. Among the topics covered in [25] we find very little on the ‘earlier’ phases, and neither [25] nor [1] offer any topic related to rigorous reasoning. This is possibly due to an underlying view of software engineering in which source code is the only ‘verifiable’ medium of expressing knowledge. In our opinion students have other courses that expose them to design and implementation issues. If we drop the requirement of full implementation in source code and instead accept specification languages as verifiable media of expressing knowledge, students can experience the development process topics and we can emphasize the ‘ontology engineering’ aspects of software engineering. Typically, the curriculum of computer science or software engineering students does not require them to take a course that would already cover these aspects.

Whether the university computing curricula in general need a stronger emphasis on complementing informal descriptions with formal ones is an issue of ongoing debate. Lethbridge’s survey has been used to argue for reducing the formal, more rigorous, forms of description [15], such as the ones described in [26]. Even where the need for a stronger emphasis on FM is accepted, there are different opinions about the way in which such stronger emphasis should be implemented in the curricula. Proposals range from elective courses on ‘logic for everybody’ [23] to the gamification of formal specification [20],¹ to an integration of relevant topics into the entire program of study that affects almost every

¹ <http://verigames.com/about-us.html>.

course related to computing or mathematics.² Bjørner proposed at least as early as 1993 (possibly earlier) this emphasis of the complementary nature of informal and formal descriptions as a cross-cutting aspect of university computing curricula in software engineering [6]. In [7], he Cuéllar expand substantially those ideas, which were later incorporated into Bjørner’s triple-volume [3–5]. Similar views are presented in [9]: “*Eventually, the working group aspires to see the concepts of formal methods integrated seamlessly into the computing curriculum so that it is not necessary to separate them in our discussions*”.

We considered the publicly available course catalog entries and syllabi of software engineering courses at numerous reputable institutions, including (but not limited to) MIT, Cornell, Stanford, and McMaster. MIT does not have an explicit software engineering course; the one that comes closest would be ‘Software Construction’,³ although it focuses most on implementation aspects. Cornell has a ‘Software Engineering’ course,⁴ which includes very little material related to formal descriptions, specifications, verification, etc. Stanford does not seem to offer any introductory software engineering course and it is not clear whether or how the relevant topics are distributed over other courses. McMaster offers many courses covering software engineering,⁵ including (but not limited to) ‘Software Design I: Introduction to Software Development’, ‘Software Design II: Large System Design’, ‘Software Design III: Concurrent System Design’, ‘Software Development’, ‘Software Requirements and Security Considerations’, and ‘Software Testing’. Only short descriptions of these courses are available on McMaster’s web pages. From the short course descriptions it is impossible to see to what extent they are concerned with the complementary nature of informal and formal descriptions.

We further considered the main undergraduate software engineering textbooks including (but not limited to) Sommerville [22], Pfleeger [19], Schach [21], and Ghezzi (et al.) [8]. All of these mention formal languages (there often called ‘formal methods’). None of them uses a suitable formal language in the cross-cutting and complementary way we would like. On the contrary, we find the treatment of ‘formal methods’ delegated to a single chapter, possibly even an online supplement.

To our knowledge Bjørner’s triple-volume [3–5] is the only textbook wherein formal text consistently complements informal text. This was our reason to select it. In his work Bjørner proposes the *triptych paradigm* which separates domain description, requirements prescription, and software design as the major phases of software development in-the-large. This paradigm differs from (e.g.) that of van Lamsweerde [14] in its identification of domain description as a phase of its own, whereas van Lamsweerde considers the elicitation and elaboration of domain properties and assumptions to be part of requirements capture. Bjørner’s

² For comparison see [18] in which such an over-arching curricular integration is proposed not for formal methods but for software security.

³ <http://web.mit.edu/6.031>.

⁴ <http://www.cs.cornell.edu/courses/cs5150/2019sp/lectures.html>.

⁵ <https://www.eng.mcmaster.ca/cas/programs/course-listing>.

books use the formal language RSL (the RAISE Specification Language) [3]: “RSL, which we primarily use in these volumes, features both property-oriented and model-oriented means of expression, has a somewhat sophisticated object-oriented means of compositionality, and borrows from CSP [...] to offer a means of expressing concurrency. Extensions to RSL have also been proposed, for example with timing [...], and with Duration Calculus, that is, temporal logic ideas [...]”.

CSP (Communicating Sequential Processes) [11] allows for describing patterns of communication and synchronization among concurrently running activities. Bjørner uses RSL and CSP throughout his entire triple-book to formally complement topics as diverse as container harbors or state machines.

3 Initial Situation

For historical reasons our institution offers several programs in computing at the undergraduate level: see Table 1.

There are 4 courses focusing on Software Engineering: ‘Introduction to Software Engineering’, ‘Software Requirements’, ‘Software Design’, and ‘Software Reliability Testing’. Moreover, there are two offers of a course called ‘Introduction to Software Engineering’: one is by the Department of Mathematical Sciences, the other by the (recently founded) Department of Computer Science and Engineering. The two courses are considered equivalent w.r.t. students’ program requirements. They only differ in the faculty assigned to the course and, as a consequence, in the textbook used, the homework assignments, and other such aspects, as they vary from one faculty member to another.

Students in all of the above programs need to take I2SE, but only students in SE need to take all the remaining courses. For students in other programs they are electives. For the students in SE and in CE the I2SE course is a pre-requisite for taking their capstone project course. The programs are ABET-accredited, except for the CS program which is currently working towards accreditation. The I2SE ABET-accredited syllabus allocates times for covering topics as shown in Table 2. When two colleagues, who had been involved in teaching I2SE, retired in

Table 1. Our undergraduate programs in computing

PROGRAM	DEPARTMENT	FACULTY/COLLEGE
Computer Engineering (CE)	Electrical Engineering	Engineering
Computer Science (CS)	Mathematical Sciences	Arts and Sciences
Computer Science & Engineering (CS & E)	Computer Science & Engineering	Engineering
Software Engineering (SE)	Computer Science & Engineering	Engineering

short succession and a new colleague started teaching the course, we considered this to be a good opportunity to take a fresh look at how I2SE is taught.

Table 2. Old I2SE: time allocated to topics

TOPIC	CONTACT HOURS
Introduction to the course	1
The Software Lifecycle	3
Estimation: Cost, effort and agenda	3
Planning and tracking	3
Risk analysis and management	2
User Interface design	1
UML language	4
Requirements analysis and specification	5
Design principles and concepts, system design testing	6
Software testing	4
Exams, discussion sessions, and presentations	13
Total: (equivalent to contact period)	45

4 Revised Course

Diverse influences contributed to the shape of the revised course. We contacted Bjørner about the use of his textbooks in our course, (see acknowledgments at the end of this paper). His books complement informal with formal descriptions in every aspect of software engineering—be it a phase of development (domain description, requirements prescriptions and software design), structuring techniques (like modularization), or the structure and semantics of computational models (like finite-state automata or Petri nets).

Example:

- A class of phenomena of *behaviors* is introduced, informally as proceeding in time by performing actions, generating or receiving events and otherwise interacting with behaviors. We then present in class (as does the textbook) as a generic simple formal example an RSL specification of two processes communicating across a shared channel and controlled by a channel each, connected to the environment. This formal example is then again complemented with text describing informally but in detail the formal text as well as a process diagram and a message sequence chart, which again informally complement the formal text.

- The first example for domain descriptions is that of air traffic. It exemplifies domain attributes we would consider continuous (in the mathematical sense) and the formal description in RSL is interleaved and complemented with informal text.
- The interface requirements for a GUI for a system allowing clients to browse time tables and staff to update such time tables likewise interleave formal and informal text.
- The textbook (and the lectures) present algebras as a foundational tool for specification and object-oriented programming is explained as an application of the algebra concept. Several examples present informal descriptions together with algebraic, formal descriptions in RSL •

We follow the complementary presentation of the textbook in the lectures. Students are reminded to complement the informal parts of their coursework with formal texts from the start of the course; later they are required to provide such complementary texts for some exercises and exam problems.

As mentioned above, Bjørner’s triple-book uses RSL as its ‘running’ formal method throughout. Additionally, the book introduces formal tools like finite state machines, push-down machines, Petri nets, message sequence charts, live sequence charts, statecharts, and CSP, which are then (with the exception of CSP, which is part of RSL) explained —i.e.: given a formal execution semantics— in RSL.

Due to the very large volume of contents provided in this triple-book, we initially *selected undergraduate-suitable material* from it in consultation with its author. As our course contributes to ABET-accredited programs, we needed to revise our course in such a manner that it remained consistent with the existing ABET-accredited syllabus. For example: for some students in programs with a capstone course requirement, I2SE would be the only course explicitly exposing them to software engineering topics. In light of these requirements we re-balanced the time spent on covering some of the triple-book’s material, but also added other material on time management, scheduling, planning and estimation, risk management, and user interfaces. These concerns, together with our ‘philosophy’ presented in Sect. 1, the choice of textbook, and discussions with colleagues finally shaped the revised course. It was first held in the 2017–2018 spring semester in 3 separate small sections with 14, 17, and 19 students, and then a second time in the 2018–2019 fall semester in one larger section with 53 students. In both cases we used Moodle as the course’s learning management system.⁶ The details changed a little between those 2 semesters, due to the difference in section size, and since we made some adjustments based on the experience from the first offering.

In our first offering we did not include time management. Also, algebra, mathematical logic, and CSP channels were discussed immediately after the chapter on the Triptych SE Paradigm (instead of somewhat later) in the course. We initially thought that students would benefit from the relatively early exposure

⁶ <https://moodle.org/>.

Table 3. Resources and topics of our new course

CONTACT HOURS	LITERATURE	TOPICS
1	[5] Ch.1	Triptych SE Paradigm
2	[12, 24]	Time management, Planning, Scheduling, Tracking
2	[5] Ch.2	Documents
2	[5] Ch.5	Phenomena and concepts
1	[3] Ch.8	Algebras
1	[3] Ch.9	Mathematical logic
2	[3] Ch.21	CSP channels
3	[5] Ch.8	Overview of domain engineering
	[5] Ch.11	Domain facets
	[5] Ch.16	Domain engineering process model
2	[4] Ch.10	Modularisation (objects)
2	[4] Ch.11	Automata and machines
1	[4] Ch.12	Petri nets
2	[4] Ch.13	Message sequence charts
1	[4] Ch.14	Statecharts
1	[5] Ch.17	Overview of requirements engineering
2	[5] Ch.19	Requirements facets
	[5] Ch.24	Requirements engineering process model
1	[5] Ch.25	Hardware-software codesign
	[5] Ch.26	Software architecture design
	[5] Ch.30	Computing systems design process model
	[5] Ch.31	Triptych development process model
1	[5] Ch.32	‘Finale’
1	[14] Ch.3.2	Risk analysis
1	(other)	User interface design
Total 29		

to these topics; however from personal conversation we concluded that placing them later in the course would benefit the students. The topics of the new course are now covered according to Table 3.

The course *emphasizes application* by giving the students *homework* for every week. We time the homework so that students have to independently study the material in order to complete the homework. The goal is to make learning more problem-based. Homework was due on each Friday before class.

For the duration of the semester, students work within a broad application domain from one of the 15 domains outlined in the textbook (assigned to them on the basis the student ID number modulo 15). We took homework exercises from

the textbook and most exercises referred to the student's domain. We chose the 15 domains from the textbook since they have already been substantially elaborated and they are each broad enough to allow students to find a distinct niche within the domain.

One of the criteria we applied when deciding on the formalisms to use in the course was the availability of *tools to support experimentation* in the formalism. We considered a long list of tools, including *eRAISE*, *rsltc*, *Overture*, *Rodin*, *Idris*, *Maude*, *Snoopy* [10], *Formose*, *jUCMNav*, *TLA+* [13], *AutoFocus3*, etc.⁷

For each tool we found some strong reason which in our opinion disqualified it as a mandatory tool in the students' coursework. For *TLA+* and *Maude* our reason was a lack of resources to prepare formal descriptions in these formalisms to accompany and replace those given in the textbook in RSL (however, see Sect. 6). We considered the treatment of concurrency in *Idris* too different to the treatment of concurrency in the textbook based on CSP. Other tools were too specialized and/or too limited.

Tool support can be a boon or a bane: if students get over tool-specific obstacles quickly enough, then support in experimenting helps them to get a better grasp of the formalism. On the other hand, tool-specific obstacles can become time-consuming and take away from time that could otherwise be spent exploring the formalism with pencil and paper. The tool support for RSL, in particular, is *not* very beginner-friendly. A case in point are syntax errors: while line and column numbers are given, no more information about the syntax rule that is being violated is available. As another example, editing statecharts in *Eclipse Papyrus* consistently aborted the entire *Eclipse* process. Based on these findings and considerations we did not mandate that students use any tool support, but we pointed from the course's Moodle page to several tools: for RSL to *eRAISE* and *rsltc*, and for Petri nets to the *Simple Petri Net Editor* and *Snoopy2*.

4.1 2017–2018: Semester 2

Since we had 3 small sections for the first offering, and no assistance, we decided on the following way of homework assessment:

- Friday's sessions were for homework discussion.
- Students individually presented one exercise at a time.
- The work was discussed with the other students.
- Students' turn followed the class list.
- If a student forfeited a turn, it was the next student's turn to present the current exercise.
- We managed an average of 5 presentations per Friday session of 50 min.
- The average of a student's best 4 presentations made the homework grade.

⁷ <https://github.com/FreeAndFair/eRAISE> <https://github.com/dtu-railway-verification/rsltc> <http://overturetool.org/> http://wiki.event-b.org/index.php/Rodin_Platform_Releases <https://www.idris-lang.org/> <http://maude.cs.illinois.edu/> <http://formose.lacl.fr/> <http://jucmnav.softwareengineering.ca/foswiki/ProjetSEG> <https://af3.fortiss.org/>.

4.2 2018–2019: Semester 1

The second offering was also without assistance. The mode of homework presentation used in the previous semester was not practical for a single group of 53 students. Therefore we decided to use peer assessment, as follows:

- The homework grade was split in two sub-grades: one for the homework a student submitted, and another one for homework a student assessed of other students.
- For every one of the 15 homework assignments we prepared a catalog of (on average) 15 assessment aspects.
- Assessment aspects had weights in the range 1–10.
- Students assessed the aspects in the range 1–100.
- Friday’s sessions (after submission) were used to discuss example solutions, to clarify remaining doubts on the assessment aspects, and to lead into the assessment in general.
- For every homework submission the submitting student assessed 5 randomly assigned peer submissions.
- Every student’s assessment had a weight of 1.
- For every homework we randomly assigned 6 student submissions to be assessed by the professor with the same assessment criteria.
- The professor’s assessments had a weight of 6.
- We used the grade calculation in the Moodle workshop activity for the grade on the assessments.⁸

5 Reception

5.1 From Students’ Blogs

One of the students’ (graded) course work activities was to regularly write a *reflective learning journal* (blog). We encouraged the students to write about any aspect of the course, its content, the presentation, their experiences, suggestions, etc. We can expect that the fact that this is graded and not anonymous will change the way the students write and also what they write. On the other hand, we get a very high response rate. We collected the following opinions from the blogs. Students stated that the course widened their understanding of software engineering as not purely programming and that they were able to apply knowledge from the class on an intern job (to their surprise). Students stated the homework was too much and they were dissatisfied when other students seemed to give them random grades in the peer assessment. Several students praised the weekly homework and assessments as very helpful for remaining invested and for deeply considering alternative solutions (by other students). One student “*hated*” the textbook and “*didn’t get*” RSL and CSP. Others considered it strange, but found it good once they got used to it. A good proportion of the course topics received praise. The topics Documents, Domain Facets, Requirements Facets

⁸ https://docs.moodle.org/36/en/Using_Workshop#Grade_for_assessment.

and Software Design, State Charts, Sequence Charts, and Petri Nets were explicitly praised by several students and deemed fun and most important topics that “*improved the modeling ability*”. The project and time management lectures were called “*impressive*” and it was suggested to extend them. One student lamented the absence of a discussion of agile methodologies (which we had mentioned), and one student appreciated the “*long exams*”.

5.2 Student Surveys

2017–2018: Semester 1. We conducted a survey towards the semester’s end. Students anonymously rated aspects concerning the professor on a scale from 1 (‘never’) to 7 (‘frequently’). Only 11 of the students from all 3 courses together responded. To summarize we list the aspects in the order of increasing average student response (on the scale 1–7): Adjusts pace of class to the students’ level of understanding (4); Explains material clearly (4.2); Stimulates interest in material (4.5); Seems well-prepared (4.9); Indicates important points to remember (5.1); Is effective, overall, in helping me learn (5.3); Provides helpful comments on homework (5.4); Indicates where the class is going (5.5); Explains thinking behind statements (5.6); Effectively directs and stimulates discussion (5.8); Shows genuine interest in students (6.2); Effectively encourages students to ask questions and give answers (6.4); Is tolerant of different opinions expressed in class (6.9); Is available outside of class (7); Treats students with respect (7).

Students also rated aspects of the course on a scale of 1 (‘no or very little’) to 7 (‘yes or very much’). Again, we list the aspects in the order of increasing average student response (on the scale 1–7): Would you likely recommend this course to a friend or fellow student? (3.1); How actionable do you think the information is that you received in the course? (4); Did the content that was delivered and the organization of the course match what you were promised in the syllabus? (5.5); How much new information and knowledge did you receive in the course? (5.6).

Finally, we asked them to provide free feedback on: What do you like best about this course? What would you like to change about this course? What do you think is this instructor’s greatest strength? What suggestions would you give to improve this instructor’s teaching? They were also asked to indicate: Approximately how many class meetings have you missed (including excused absence)?

Respondents to these ‘open’ questions mentioned that they liked the weekly homework, the clarity of presentation, feeling that they became better organized in problem solving in general, as well as an improvement in their presentation skills. The weekly homework with the student presentations was mentioned most.

On the other hand, respondents would like to change the textbook, the grading of the homework (expressing that not presenting when they had done the homework felt like a waste of time), and to add more coverage of RSL. Respondents also suggested changing the homework evaluation (again expressing that not presenting when they had done the homework felt like a waste of time),

shortening the exams and switching to a different formal language than RSL. Changing the homework evaluation was mentioned most.

2018–2019: Semester 2. We conducted a similar survey after the second offering of the course. The response rate was lower than the first semester. This is likely due to the fact that one student created his own survey during the semester. Thought we know the results of this survey, too, the student's own survey was structured very differently from ours which makes the results hard to compare.

Only 6 of the 53 students responded to our survey. Again we summarize their responses concerning the professor by listing the aspects in the order of increasing average student response (on the scale 1–7): Adjusts pace of class to the students' level of understanding (3.2); Is effective, overall, in helping me learn (3.5); Stimulates interest in material (3.7); Explains material clearly (3.8); Effectively directs and stimulates discussion (4.0); Provides helpful comments on homework (4.0); Indicates important points to remember (4.2); Explains thinking behind statements (5.0); Effectively encourages students to ask questions and give answers (5.0); Is tolerant of different opinions expressed in class (5.2); Seems well-prepared (5.2); Shows genuine interest in students (5.3); Indicates where the class is going (5.5); Is available outside of class (5.7); Treats students with respect (6.2).

As far as the course as such was concerned: Would you likely recommend this course to a friend or fellow student? (2.7); How actionable do you think the information is that you received in the course? (3.0); How much new information and knowledge did you receive in the course? (3.7); Did the content that was delivered and the organization of the course match what you were promised in the syllabus? (4.5).

In the 'open feedback' section respondents liked the practicality of the topics, their exposure to project planning, that "*it teaches you good problem solving skills*", and their experience in peer assessment.

On the other hand, they disliked the slides from the textbook and would like to see the use of RSL removed, or at least more time for its introduction. They would also like to change the domain, to which they were assigned for the semester, to be more like a project they pursue throughout the semester.

6 Lessons Learned, and Future Work

The blogs (high participation, not anonymous) and the surveys (low participation, anonymous) by themselves give very incomplete pictures. Even when they are combined we have to be careful not to follow a few opinions just because they are loudly voiced.

Respondents Did Not Find the Course's Message Very Actionable.

For the future, we hope to improve this by more strongly tying the homework exercises together into a single project. While the textbook already goes a long way towards this end, we feel that students' perception of the exercises and the perceived actionability of their experience will improve with this change.

Respondents Disliked the High Workload of the Homework.

We hope that this perception changes at least in part when we integrate (most of) the current individual homework exercises into one semester-long project.

No Respondent Expressed a Like for RSL, Some Expressed Dislike, Others Did Not Mention It.

We need to evaluate how to best improve this. Options include spending more time introducing RSL at the start of the course. This would clearly help students with reading and writing in that language. It would not address students' concern that there is too little additional material (tutorials or blogs) available, and/or no visible use of RSL in the IT industry. Another option would be to switch to a different formal language, (e.g. TLA+/Pluscal). While this would give the students many more secondary sources of material to study, it would also require a thorough evaluation of the differences (e.g. in the representation of internal vs. external nondeterminism, and in the treatment of concurrency and synchronization), as well as a redevelopment in TLA+/Pluscal of a substantial portion of the examples used in the textbook.

Dislike of the Slides Accompanying the Textbook.

An obvious option would be to create slides in a style similar to the slides on time management, scheduling, tracking, and estimation, which the respondents preferred. We also consider transitioning more and more to a 'flipped classroom', which implies successively replacing slide-based lectures with in-class demonstrations of solving example problems.

Lacking Adjustment of Pace to Students' Level of Understanding.

The textbook is a great source of knowledge and wisdom on software engineering. Possibly, the level of detail is too much for our audience, and we would do better limiting the lectures to the most prominent parts of the messages of the textbook.

Lacking Clarity of Explanation.

Here also, we might see an improvement by focusing the message of the lectures on fewer statements from the voluminous textbook.

Lacking Stimulation of Interest in Material.

We will have to analyze this result much more to understand the reasons for failing to stimulate interest in the past and how to succeed in this aspect in the future.

7 Conclusion

We have shared our experience about revising our I2SE course to reflect better what is (in our opinion) a more modern approach to teaching software engineering which is rooted in our philosophy that formal texts and informal texts must complement one another. We explored existing teaching materials and courses offered elsewhere on whether they were compatible with our philosophy. Except for the textbook we chose, none of the others seemed compatible with our views. Our initial situation imposed diverse constraints on the course's revision. We had

to adjust some of our initial choices in order to satisfy these constraints. The course now differs considerably from traditional I2SE courses in its use of suitable formalisms to complement most of the topics of traditional I2SE courses. Reception is still far from what we aim for. In future offerings of the course we will make improvements based on the feedback we received from our past students.

Acknowledgments. We sincerely thank the three anonymous reviewers of SACLA' 2019 for their critical reading of earlier versions of this paper and for their suggestions which helped to improve and to clarify it. Many thanks also to *Dines Børner* for our valuable conversations about this topic.

References

1. Abelson, H., Greenspun, P.: Teaching Software Engineering (2001). <http://philip.greenspun.com/teaching/teaching-software-engineering>
2. Baber, R.: Comparison of electrical 'Engineering' of Heavyside's times and software 'Engineering' of our times. *IEEE Ann. Hist. Comput.* **19**(4), 5–16 (1997)
3. Bjørner, D.: Software Engineering 1: Abstraction and Modelling. Springer, Heidelberg (2006). <https://doi.org/10.1007/3-540-31288-9>
4. Bjørner, D.: Software Engineering 2: Specification of Systems and Languages. Springer, Heidelberg (2006). <https://doi.org/10.1007/978-3-540-33193-3>
5. Bjørner, D.: Software Engineering 3: Domains, Requirements, and Software Design. Springer, Heidelberg (2006). <https://doi.org/10.1007/3-540-33653-2>
6. Bjørner, D.: University curricula in software technology. In: Software Engineering Education, pp. 5–16. Elsevier (1993)
7. Bjørner, D., Cuéllar, J.R.: Software engineering education: rules of formal specification and design calculi. *Ann. Softw. Eng.* **6**(1/4), 365–409 (1998)
8. Ghezzi, C., Mehdi-Jazayeri, D.M.: Fundamentals of Software Engineering, 2nd edn. Pearson Prentice Hall, Englewood Cliffs (2003)
9. Goelman, D., Hilburn, T.B., Smith, J.: Support for Teaching Formal Methods: Report of the ITiCSE 2000 Working Group on Formal Methods Education (2000). <http://www.cs.utexas.edu/users/csed/FM/work/final-v5-7.pdf>
10. Heiner, M., Herajy, M., Liu, F., Rohr, C., Schwarick, M.: Snoopy – a unifying Petri net tool. In: Haddad, S., Pomello, L. (eds.) PETRI NETS 2012. LNCS, vol. 7347, pp. 398–407. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-31131-4_22
11. Hoare, C.A.R.: Communicating Sequential Processes. Prentice Hall, Englewood Cliffs (1985)
12. Humphrey, W.: Introduction to the Personal Software Process. Addison Wesley, Reading (1997)
13. Lampert, L.: Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers. Addison Wesley, Reading (2002)
14. van Lamsweerde, A.: Requirements Engineering: From System Goals to UML Models to Software Specifications. Wiley, Chichester (2009)
15. Lethbridge, T.: What knowledge is important to a software professional? *Computer* **33**(5), 44–50 (2000)
16. Mills, B.: Practical Formal Software Engineering: Wanting the Software You Get. Cambridge University Press, Cambridge (2009)

17. Motta, G., Wu, B. (eds.): *Software Engineering Education for a Global E-Service Economy: State of the Art, Trends and Developments*. Springer, Cham (2014). <https://doi.org/10.1007/978-3-319-04217-6>
18. Ngwenya, S., Fitcher, L.: A framework for integrating secure coding principles into undergraduate programming curricula. In: Tait, B., Kroeze, J., Gruner, S. (eds.) *SACLA 2019, CCIS*, vol. 1136, pp. 50–63. Springer, Cham (2020)
19. Pfleeger, S., Atlee, J.: *Software Engineering: Theory and Practice*. Pearson Prentice Hall, Englewood Cliffs (2006)
20. Prasetya, I.S.W.B., et al.: Having fun in learning formal specifications. Technical report (2019). <http://arxiv.org/abs/1903.00334v1>
21. Schach, S.: *Object-Oriented and Classical Software Engineering*. McGraw-Hill, New York (2011)
22. Sommerville, I.: *Software Engineering*. Pearson, London (2011)
23. Spichkova, M.: Boring formal methods or Sherlock Holmes deduction methods? Technical report (2016). <http://arxiv.org/abs/1612.01682v1>
24. Spolsky, J.: Evidence Based Scheduling (2007). <https://www.joelonsoftware.com/2007/10/26/evidence-based-scheduling/>
25. Vallino, J.R., Basham, B.: A re-look at the introduction to software engineering course. In: *Proceedings of ASEE Annual Conference and Expos* (2018)
26. Zamansky, A., Farchi, E.: Exploring the role of logic and formal methods in information systems education. In: Bianculli, D., Calinescu, R., Rumpe, B. (eds.) *SEFM 2015. LNCS*, vol. 9509, pp. 68–74. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-49224-6_7



Exercise Task Generation for UML Class/Object Diagrams, via Alloy Model Instance Finding

Violet Kafa, Marcellus Siegburg, and Janis Voigtländer^(✉)

University of Duisburg-Essen, Duisburg, Germany
janis.voigtlaender@uni-due.de

Abstract. The Unified Modelling Language (UML) is the standard for designing and documenting object-oriented software systems. Its most frequent use is for static modelling in the form of class diagrams. A correlated concept is that of object diagrams. An object diagram may or may not adhere to a given class diagram, and the understanding of this connection is key to correctly using class diagrams in practice. We present an approach for automatic generation of verified, non-trivial, conceptually relevant examples and counterexamples of class/object diagram combinations, aimed at providing exercise tasks in a university course setting. The underlying technique is model instance finding using the Alloy specification language and analyser. We provide an implementation of our approach in an e-learning system.

Keywords: E-learning · UML · Alloy

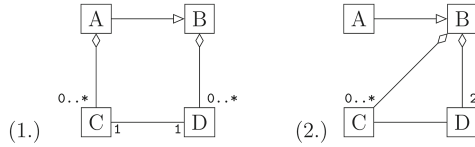
1 Introduction

The Unified Modelling Language (UML) [10] is widely used in the software industry and academia. It is a standard for specifying, visualising, constructing and documenting artifacts of object-oriented systems and has a rich set of diagrammatic notations along with their well-formedness rules. The language of class diagrams (CDs) and object diagrams (ODs) is part of the UML standard and supported by many commercial and academic software modelling tools. These specific diagrams are also a typical subject matter in a software modelling course at university.

To facilitate learning and understanding of the CD and OD concepts, it is desirable to confront students with many and diverse examples and counterexamples. For instance, a useful exercise task in a software modelling course is to present a certain number n of CDs and a certain number m of ODs and ask students to determine for each combination of CD and OD whether the latter is a correct instance of the former or not, along with explanation of the reasons (such as possible violations of multiplicities or other constraints). Fig. 1 shows a hand-crafted exercise task of this kind, with $n = 2$ and $m = 5$, used in a concrete course in the past. Our undertaking here is to develop tooling that helps

the instructor by systematically and automatically constructing similar exercise tasks.

Let the following class diagrams be given, each of which shows connections between the classes A, B, C and D.



Indicate, for each of the following object diagrams, whether it is valid for the above class diagrams (ten answers altogether). Where that is not the case according to you, explain why and give all reasons.

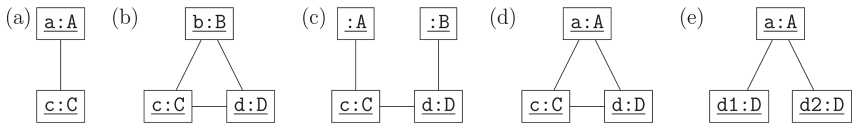


Fig. 1. A sample exercise task.

To that end, formal method techniques from the verification and model generation domain are employed. The basic idea is to randomly generate CDs subject to certain complexity constraints, and with a reduced feature set according to didactic considerations, and to use the Alloy specification language and analyser [5, 6] to generate appropriate model instances and non-instances, to be presented as candidate ODs to the students. To make use of Alloy, we employ (and extend/revise) a translation from CDs to Alloy modules that was introduced in related work on CD analysis and OD generation [8]. For the generation of interesting counterexample ODs, which was not a topic of the mentioned CD2Alloy work, we devise a strategy that involves variations of the original CD, such as removing or adding some relationships, manipulating some multiplicities etc.

2 Background on UML’s CDs and ODs

There is a vast literature on UML, including many textbooks introducing its various diagram types [2, 4, 11], so we will not provide another substantial introduction here. But we want to at least give motivating examples of a CD and a conforming OD, to illustrate which model elements these types of diagrams can contain, as well as to shortly discuss the relevant relationships. Moreover, we already briefly delineate what aspects we will *not* cover in our generated exercise tasks (more details then in the next section, about didactic considerations).

Fig. 2 shows a CD, illustrating the static design of a certain object-oriented system. In it, we see classes, some with attributes and operations/methods, an inheritance relationship, and additional relationships in the forms of compositions and general associations with attached multiplicities.

Fig. 3 shows an OD conforming to the CD from Fig. 2. Note that no operations/methods are present in ODs. In our exercise tasks we will actually cover neither attributes nor operations/methods, since we are more interested at this point in teaching in considering the relationships between classes/objects.

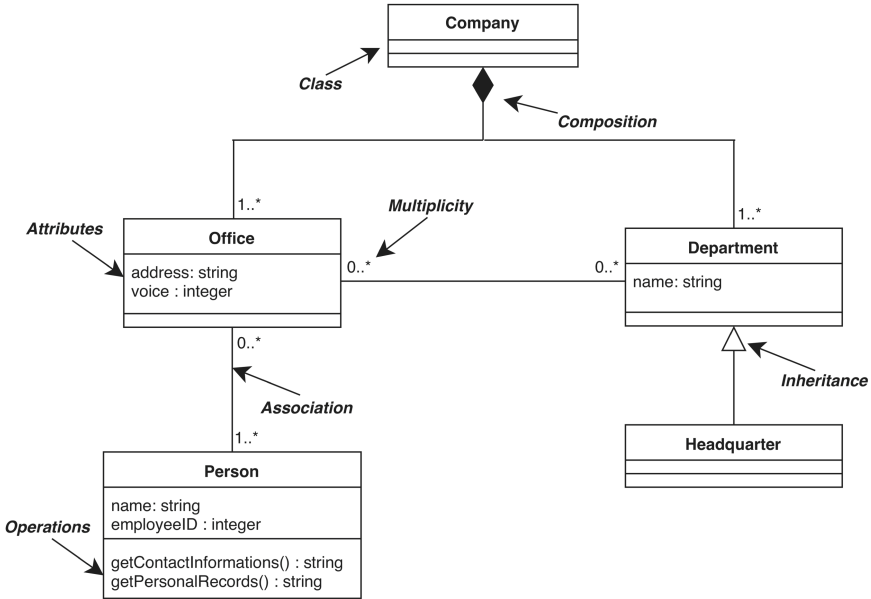


Fig. 2. A CD.

Inheritance between classes is not reflected on the object level by explicit connections/links between corresponding objects in any way. Instead, inheritance expresses that the child class has all the same relationships to other classes as the parent class has, and *that* will be reflected on the object level. For example, according to Fig. 2 every headquarter is a department, so in Fig. 3 we could have replaced **d1** by an object of type **Headquarter** and still let it have the link to **c**.

Association between classes, such as between offices and persons or between offices and departments in Fig. 2, is a broad term that encompasses just about any logical connection between classes. On the object level, associations are represented by links, such as between **o** and **p** in Fig. 3. Multiplicities at the ends of associations in a CD express how many objects of one class can be related to one object of the other class. In our example, each office needs to be linked to at least one person (due to the multiplicity 1..* at one end of the relevant association), but not each person needs to be linked to an office (due to the multiplicity 0..* at the other end). So in Fig. 3 we could not simply delete **p** (while keeping **o**), but we could add another object of type **Person** without adding any links. We consider 0..* the default multiplicity, so it will not always be depicted (see Fig. 1).

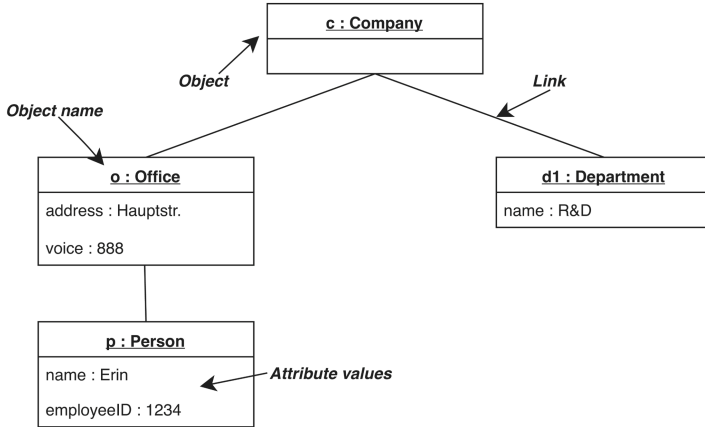


Fig. 3. An OD.

Another kind of relationship is aggregation. It does not appear in Fig. 2, but in Fig. 1, see the hollow diamond shapes there. While from a modelling perspective aggregations are special, in that they are intended to represent whole/part relationships, from a formal perspective concerning conformance of ODs to CDs, aggregations are to be handled just like associations. Their default multiplicities are also as for associations.

Finally, a composition relationship is a stronger form of aggregation which actually comes with additional constraints in considering conformance of an OD to a CD. Namely, every “part” must be linked to at most one “whole”. The example in Fig. 2 has a composition relationship between companies and offices, and another one between companies and departments, with companies playing the role of “whole” in both cases (as marked by the filled diamond shape). So in Fig. 3 there could be additional companies, but they could not also be linked to the existing objects **o** and **d1**. Instead, there would have to be additional offices and departments present, due to the multiplicity 1..* at the “part” end of both composition relationships. The default multiplicity at those ends would have again been 0..*, while at the “whole” end of composition relationships the default multiplicity is 1..1 (which would be written simply as 1) and actually only the multiplicities 0..1 and 1..1 are allowed there at all.

Besides the fact that we will not include attributes or operations/methods in our generated examples (manifesting in depictions as in Fig. 1, each class and object being just a simple box with name and/or type inscribed, instead of additional compartments in the boxes as in Figs. 2 and 3), another difference from what we have seen in the current section is that the generated examples will be artificial, with class and object names like **A**, **B**, **c**, **d**, instead of real world notions like **Company** etc. The reason is that we want to use the generated exercise tasks for teaching the formal concepts of the CD and OD language, and the correct interplay between model elements, emphasising the similarities

and differences between the relationship flavours considered. Thus enabled, free modelling set in real world scenarios is part of separate activities in the course.

3 Didactic Considerations

What makes a good generator for exercise tasks of the kind considered? We would like to be able to generate many different, but analogous tasks, in order to provide students with ample opportunities for practising without repeated or predictable instances/solutions. We want to be able to control the complexity and difficulty of tasks, for example to enable a transition from simple, preparatory instances to more challenging ones, or to level the field for student groups from different backgrounds (e.g., ones that have already had an object-oriented programming course and thus know some UML concepts at least from a programming language perspective, and ones for which this is not the case). There are additional dimensions along which we would like to be able to parametrise the task generation. For example, at a certain teaching stage we might want to tailor tasks to focus on one specific concept (“give us only tasks in which the correct understanding of aggregation makes the difference between right and wrong, despite other relationships also being present”) or to exclude some concept (“do not produce tasks with occurrences of composition relationships, because we have not yet covered that in the lecture”). Generated tasks should be non-trivial and interesting, in the sense that there is really something to discover in them. For example, if the task is to decide for a CD and two ODs which one of the latter two conforms to the CD and which one does not, then it should not be the case that one of the two ODs is obviously far off from possibly having anything to do with the given CD, thus making the question boring and uninteresting. Instead, we should have a degree of control over how far off a counterexample OD is allowed to be from a positive example. And while it might be obvious, it is crucial to ensure that the generated exercise tasks are correct. If we consider a certain OD to be (or to not be) conforming to a certain CD, and use that assessment in feedback to students, or for grading, then it better be the case that it holds true. When hand-crafting artificial tasks that try to emphasise a certain CD/OD language aspect as well as aiming to produce interesting and challenging instances, it can be surprisingly easy to violate this assurance by accidentally not taking into account some subtlety of the UML standard.

Besides the above general considerations, there are more concrete decisions to make about the design of the exercise tasks (generator). For example, in Figs. 1, 2, and 3, we have not annotated names for associations, aggregations, and compositions. UML does allow such annotations, and sometimes they – or some other means of distinguishing links – are needed to properly decide about conformance between CDs and ODs. As a simplistic example, consider the two CDs on the left in Fig. 4 and assume we want to provide an OD that conforms to the first CD, but not to the second CD. The OD on the left in Fig. 5 fits the bill. But if we omit association names, see the right halves of Figs. 4 and 5, this is not discernible anymore, because now the link between objects a and b could be accidentally perceived as stemming from the association between A and C

that B inherits in the second CD. The examples in Fig. 1 were carefully crafted to ensure that no such confusion exists, or put differently, that students always have a chance to puzzle out which association or aggregation link corresponds to, even in the absence of names. But for our task generator we have decided to always provide those names, thus making the generated tasks more beginner friendly in that respect.

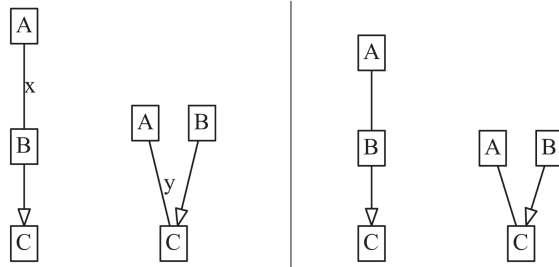


Fig. 4. CD examples with and without association names.



Fig. 5. OD examples with and without association names.

On the other hand, we do omit other annotations on associations, aggregations, and compositions: namely role names and navigation or reading directions. This also has an impact on the level of challenge the checking of conformance poses. For example, if we present the CD shown on the left in Fig. 6, then students are likely to be sceptical about conformance of an OD that has two y links between the same two objects of type A, as in the middle of Fig. 6. After all, even though A inherits from B and is thus allowed (since B is) to be linked with A via y, the double linkage seems strange and at least subtly at odds with the multiplicities written on the ends of y in the CD. If, however, we were to annotate direction arrows on the associations in the CD, and then present the OD also with direction arrows, as on the right in Fig. 6, then the situation would be less surprising (since the revelation “the two y-links go in opposite directions” makes things clearer), and thus possibly less of a challenge. The point here is, even with names provided on associations, aggregations, and compositions, there are still interesting bits to puzzle out by the learner.

We have made further decisions about the CDs and ODs to generate. For example, we do not allow multiple inheritance and we impose structural constraints

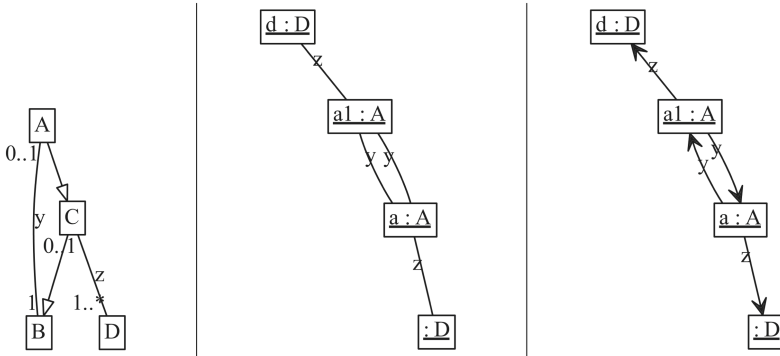


Fig. 6. A somewhat surprising case of conformance.

(such as no two associations between the same pair of classes in a CD) that are not mandated by the UML standard. These decisions are also driven by what aspects we want the learning to focus on, by experience with hand-crafted exercise tasks, and by trying to balance challenge and approachability of tasks. We might revise them, or the decisions about providing association names and/or other annotations, as we gain experience with automatically generated tasks. In any case, even within the frame of the decisions as currently set, we have enough means to produce tasks of varying difficulty, such as by parametrisation over the numbers of classes, inheritances, associations, . . . , objects, links.

4 Background on Alloy Usage

It is important that the exercise tasks we will generate automatically, and present to students without prior inspection of every instance by an instructor, are one hundred percent semantically correct. If the e-learning system assesses that a certain OD does, or does not, conform to a certain CD, then that should be guaranteed. Simply programming a random generator for CD/OD pairs that should serve as examples and counterexamples in an exercise task would risk introducing errors, even under best effort to truly implement what the UML standard implies. Moreover, using such a hand-written coupled generator, it would probably be difficult to tailor generated tasks for special, possibly changing needs, such as emphasis on certain model elements and diagram features. So instead we decided to take a more declarative, and at the same time verifiable, route. We build on the CD2Alloy work [8] that translates CDs to modules in Alloy, a specification language based on mathematical sets and relations and backed by SAT solving [6].

Compared to other translations from the UML domain to Alloy [1, 9, 12] (and in reverse, to interpret models found by the SAT solver back into UML as object diagrams), the CD2Alloy translation performs a deeper embedding of CD concepts into the Alloy logic. That is, instead of rather directly mapping CD concepts (classes, associations, . . .) to quite-similar-but-not-really-equivalent Alloy

concepts (signatures, fields, ...), the translation in some sense explicitly programs out the UML semantics as functions and logical predicates. That allows more accurate representation, capture of more UML features, and very importantly, analysis over more than one CD at a time. In particular the last aspect will be crucial for us here. The details of the embedding/translation are not of superior importance for the current work, but to at least provide a flavour, Fig. 7 shows excerpts of the Alloy module obtained for the CD shown on the left in Fig. 6. For example, the `fun` definitions essentially express that in any model the set of objects of type A is exactly the set of objects directly belonging to A (since A has no subclasses in the CD), while for example the set of objects of type B is the union of the sets of objects of B, C, and A (due to the inheritance relationships in the CD). And the lines involving `ObjLUAttrib` and `ObjLU` express the multiplicities at the two ends of association y in the CD, using predicates whose definitions are not shown in the excerpt. The translation rules are described in a technical report [7]. For our purposes here we use a subset of them (due to our restricted set of CD features used), and actually had to perform a few revisions/adaptations (to more exactly express some desired constraints). But the fundamental principles are as in the previous work.

```

...

    one sig y extends FName {}

...

    fun ASubsCD : set Obj {
      A
    }
    fun BSubsCD : set Obj {
      B + CSubsCD
    }
    fun CSubsCD : set Obj {
      C + ASubsCD
    }

...

    ObjLUAttrib[ASubsCD, y, BSubsCD, 1, 1]
    ObjLU[BSubsCD, y, ASubsCD, 0, 1]

...

```

Fig. 7. A glimpse of Alloy code for the CD from Fig. 6.

An Alloy module as in Fig. 7 (completed) can be given to the Alloy analyser, which will try to find a model/instance, in a finite scope, and will return any ones existing in a textual form. That textual form can be interpreted as

describing ODs. In fact the ODs in the middle and right (depending on whether navigation directions are to be depicted or not) of Fig. 6 are thus obtained. An Eclipse plug-in exists that implements this whole workflow, thus for example allowing a software engineer to check whether a certain software design is meaningfully populated at all (by letting ODs be created for inspection, from the CD of a software system/component under development). Since our objective here is different, we need a different workflow/approach (see next section), but the CD2Alloy translation is a crucial ingredient.

It is worth pointing out that we are still using a hand-written random generator, but not for CD/OD pairs, just for CDs. That is, CDs are not generated by the Alloy analyser from some meta model. Instead, we have programmed a CD generator (see description in Sect. 6) that is driven by our didactic considerations about which features to support, which structural constraints (beyond those mandated by the UML standard) to adhere to, etc., see also relevant discussion in the previous section. That was a pragmatic decision and does not incur the risk painted in the first paragraph of the current section, concerning semantic correctness. After all, creating CDs essentially just means we need to get the syntax right. The semantics will be covered (and verified!) via Alloy.

5 Approach to Counterexample Generation

Being able to provide positive instances of ODs for a given CD is nice, but only at most half (actually much less) of what we need for our exercise task generation. We also need to be able to provide negative instances, counterexamples. One tempting approach, given the deep embedding of CD concepts into Alloy logic that CD2Alloy performs and which allows first-class use of all of Alloy's logic expressivity on top, would be to simply make use of logical negation. That is, given that the OD in Fig. 6 was obtained (along with many others) by calling the Alloy analyser on the module from Fig. 7 with command:

```
run { cd } for 4 Obj
```

where `cd` is the name of a predicate that builds on all the things introduced by the translation for the given CD (such as the `fun ASubsCD` definitions etc.), and where the `4` stands for at most how many objects should be present in the OD, a naive attempt at counterexample generation would be to instead call the Alloy analyser on the same module but with command:

```
run { not cd } for 4 Obj
```

Unfortunately, the results are of questionable value. Fig. 8 shows the OD-style rendition of the first of a multitude of instances found. That is indeed not an OD conforming to the CD from Fig. 6, but it is utterly useless for an exercise task, because it is off at first look. In fact, even if we were to somehow structurally constrain or filter out instances that seem too far off from being worth showing as solution candidates in an exercise task, for example by discarding everything

with perceivedly too many parallel edges, we would not necessarily be better served. Specifically, without additional precaution there lurks a potential deeper problem here, because the predicate `cd` conceptually expresses:

“[what Alloy finds as model] is an OD instance of the given CD”

and the logical negation of that (in formal or in natural language) happens to not be:

“is an OD that is not an instance of the given CD”

but instead simply:

“is not an OD instance of the given CD”

which actually means:

“is not an OD or is an OD that is not an instance of the given CD”.

So it is not ruled out up front that `not cd` might produce structures that are not even proper ODs. We could dispel this concern by additional analysis, but that does still not necessarily, generally give us instances that would serve as *useful* counterexamples in an exercise task.

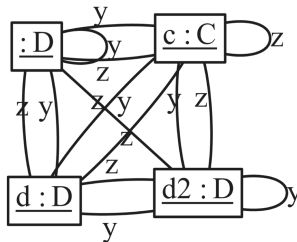


Fig. 8. Result of a naive attempt at counterexample generation.

Instead, we make use of a crucial advantage of the CD2Alloy translation/embedding over competing approaches, namely its ability to perform analysis over more than one CD at once. This ability was already briefly mentioned in the previous section and comes in handy now. The basic idea is that if we have a CD and want to produce a counterexample OD for it, we could look for a structure that is not just *not* an OD for *that* CD, but at the same time actually *is* an OD for a CD *similar* to, but somewhat mutated from, the original CD. That way, we can guarantee that we are not getting something wildly off, and we can even control in some sense how much and which kind of conceptual distance there will be between OD examples and counterexamples we present, by controlling the degree and character of mutation applied on the CD level. For example, if we want to emphasise/train the semantics of composition relationships, we could take an original CD that contains a composition, produce a new CD via a

mutation that specifically affects this composition (moving one of its endpoints to a different class, changing the multiplicity at one of its ends, or even turning the composition into an aggregation or other relationship), and then look for an OD that is an instance of the mutated CD, but not of the original CD. What we will get is an OD that is a counterexample for the original CD, and is so not by having nothing to do with that original CD at all, but instead more targetedly by something having to do with the composition relationship we touched. Of course, we would not tell the students that touching this composition relationship in the way we did is what happened in the background, in order to not disclose too much about what they should actually discover in the example/counterexample.

So our approach in its simplest form can be described as follows. We have a CD (randomly generated), and translate it into an Alloy module. The relevant all-encompassing predicate resulting from that could be called `cd1`.¹ We mutate the original CD to a similar one, and put that one through the `CD2Alloy` translation as well, resulting in an overall predicate `cd2`. By combining Alloy modules appropriately (there is some overlap between modules due to common definitions, so combining does not simply mean concatenating), we obtain a single one for which we can call verification commands like:

```
run { (not cd1) and cd2 }
```

or:

```
run { cd1 and (not cd2) }
```

Instances found by the former call correspond to relevant OD counterexamples for the original CD, as outlined in the previous paragraph. And instances found by the second call can also be used: as interesting positive examples for the original CD, or of course as counterexamples for the mutated CD, or indeed as part of cross check exercise tasks like the one in Fig. 1.

The approach as described above is sketched in Fig. 9. Actually, we use a more elaborate strategy, to be discussed in the next section. Here, let us just note that it is not hard to imagine that the combination approach generalises well to more than two CDs, so that we can consider instances that have certain positive or negative inhabitation properties regarding three or more CDs. That will be used to create more interesting exercise tasks.

As a side note, an alternative approach could conceivably have consisted of not mutating a CD, for which one or more conforming ODs are already given and counterexample ODs are still being sought, but instead performing mutation on the OD level, that is, turning a conforming OD into a counterexample OD by deleting a link or similar changes. However, we would then still have to check whether the obtained OD has all the desired characteristics relative to the given CD(s). After all, deleting a link may or may not turn an example into a

¹ That involves variants of the stuff shown and elided in Fig. 7, e.g., definitions `fun ASubsCD1` etc., while for the predicate `cd2` considered in a moment, separate variants `fun ASubsCD2` etc. would be produced.

counterexample. So after applying OD mutations we would have to separately establish that what we got is an OD that conforms to the CD(s) we want it to conform to and does not conform to the CD(s) we want it to not conform to. And if it turned out that we did not get what we wanted, we would have to try some other changes. Instead of such a search-and-check procedure, our declarative approach (expressing the desired characteristics in Alloy verification commands) is more targeted and directly gives us appropriate ODs, if they exist.

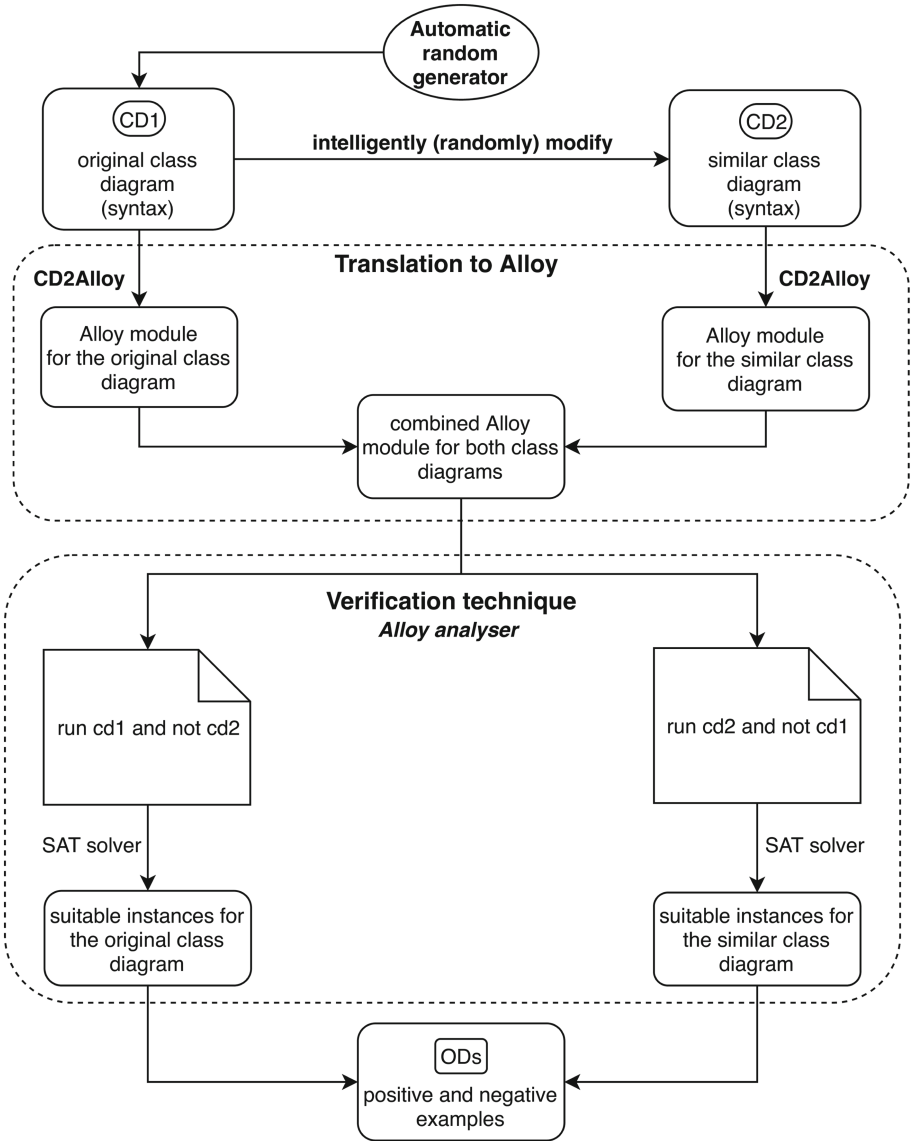


Fig. 9. Sketch of the approach in its simplest form.

6 Strategy and Implementation

As already mentioned, we are using a hand-written random generator for CD syntax without attributes or methods. The generation is parameterised by a configuration comprised of a minimum and maximum number each for classes, inheritance relationships, associations, aggregations, and compositions. The default configuration is: exactly four classes, between one and two inheritances, at most two associations, at most two aggregations, and at most one composition. The multiplicities used in the CD are drawn from a small set of choices: the default multiplicities and some special cases like 0..1, 0..2, 1..*, but not arbitrary $n..m$. We do not create CDs with multiple inheritance (one class having two outgoing inheritance arrows) and impose some additional structural constraints:

- Classes have no self-relationships of any kind. That is, a class cannot inherit from itself, cannot have an association relationship to itself, etc. Note that this does not mean that objects cannot have self-links, in fact we will see the opposite at the end of this section.
- There is at most one relationship between the same pair of classes.
- There are no inheritance cycles and no directed composition cycles.

Some of these are actually already imposed by the UML standard, which we adhere to anyway and also beyond what is listed above.

When it comes to mutating a CD to another one, we randomly choose from the following operations (while preserving all constraints mentioned above):

- adding a new relationship between any two classes,
- removing an existing relationship,
- flipping the direction of an existing inheritance, aggregation, or composition relationship,
- exchanging an existing relationship with another relationship, e.g., turning an inheritance into an aggregation,
- changing an existing multiplicity by increasing, decreasing, or shifting its range.

Note that some of these have the character that a CD2 obtained from a CD1 can only ever have more (or only ever have fewer) OD instances than originally. In such cases, simply using `run {cd1 and not cd2}` and `run {cd2 and not cd1}` as sketched in Fig. 9 would not result in interesting exercise tasks, since at most one of these commands would actually produce instances. Possibilities to counter this include to perform more than one mutation and/or to involve more than two CDs. Another reason to involve at least a third CD is that we would also like to present ODs in an exercise task that do not conform to either of two given CDs.² But simply calling `run {not cd1 and not cd2}` is not a good idea, due to the same issues that led to Fig. 8. So a third predicate `cd3` should

² For example, can you spot which of the five ODs in Fig. 1 do not conform to either of the two CDs given there?

be involved, for yet another mutated CD. Actually, our exercise task generation strategy uses overall four CDs, and is described next.

We step through an explanation of our generation strategy in the remainder of this section, along with a concrete example. First of all, we create a random CD, see CD0 in Fig. 10. In this case, it so happens that CD0 does not contain any non-inheritance relationship. That limits which mutations are applicable in the next step here, since for example there are no multiplicities to change, but in general the next step chooses, twice, from the whole assortment of mutations listed above. We mutate CD0 into CD1, and CD0 into CD2, see again in Fig. 10. In this case, CD1 was obtained by adding an association, and CD2 by adding an aggregation elsewhere. The thick edge in CD1 will be explained in a short while; to students it will be shown as normal edge. In general, we now have CD1 and CD2 that are one or two mutations apart from each other.³ We only continue if at least one of them contains a non-inheritance relationship; otherwise we start over with generating a new CD0. The motivation is that CD1 and CD2 will be the CDs included in the exercise task, and having them contain only inheritance relationships would not give interesting tasks. Next we mutate CD0 yet again, into CD3, see Fig. 10. This time, an inheritance was turned into an aggregation.

Now, Alloy is asked to find instances for all combinations of CD1/CD2 satisfied positively/negatively, with CD3 involved as a “safeguard” in the otherwise all negative case:

- cd1 and not cd2
- cd2 and not cd1
- cd1 and cd2
- not cd1 and not cd2 and cd3

The search is limited by a maximum number of objects allowed, our default being four (see earlier remarks on configurability). In addition, structural constraints on the ODs are possible. At the moment, we use the following Alloy code:

```
fact LimitIsolatedObjects {
    #Obj > mul[2, #{o : Obj | no o.get and no get.o}]
}
```

to prevent generation of instances in which half or more of the objects are not linked to other objects.

In the concrete example, the numbers of OD instances found for the calls listed above are: 23, 9, 0, 5. So here there exists no OD (within the search constraints) that is an instance of both CD1 and CD2. From all the ODs we now randomly choose five, while ensuring that not more than two are taken from the same “bucket”, i.e., not three or more ODs that satisfy `cd1 and not cd2`, etc.

³ By happenstance, they could also be identical, but that would be detected and rejected in a later step.

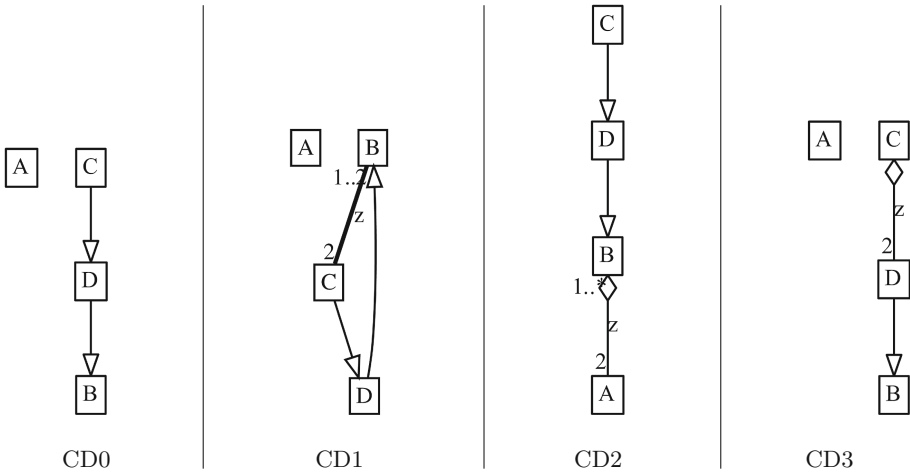


Fig. 10. CDs created in the concrete example.

If that is not possible, we start over with generating a new CD0.⁴ Fig. 11 shows the five ODs obtained in our example run of the generator, each annotated with the bucket it came from.

We have implemented the strategy explained above, along with visualisation etc., and made it available through integration into an instance of the Autotool [13] e-learning system at <https://autotool.fmi.iw.uni-due.de/alloy-cd-od>. What is shown to students for each task are CD1 and CD2 (but thick edges turned normal) and OD1–OD5, of course without the bucket annotations. Immediate feedback is provided on student answers by checking them against what the system knows about the origin (buckets) of the presented ODs. Seeds for the random task generator would be derived from student identification numbers in an actual course.

What remains to be done here in the paper is to explain the role of thick edges in CDs. These are associations, aggregations, or compositions that interact in a somewhat subtle way with inheritance. For example, the thick association in CD1 in Fig. 10 will be inherited at one end from B to D to C, with the consequence that C objects can have links to themselves (see OD3 in Fig. 11). Ultimately, the “puzzle” concerning Fig. 6 in Sect. 3 was also caused by a “thick edge” (though it was not depicted thick there). So treatment of these specific constellations of relationships in CDs is one way of making exercise tasks less or more challenging. We currently permit them for CD0, for at most one of CD1 and CD2, but not for CD3. We do not disclose their presence or

⁴ This is also the step where we would reject the case that CD1 and CD2 happened to be identical. For if they were, then the first two buckets, **cd1** and **not cd2** as well as **cd2** and **not cd1**, would be empty, and it would be impossible to choose five ODs from the remaining two buckets while not taking more than two from one bucket.

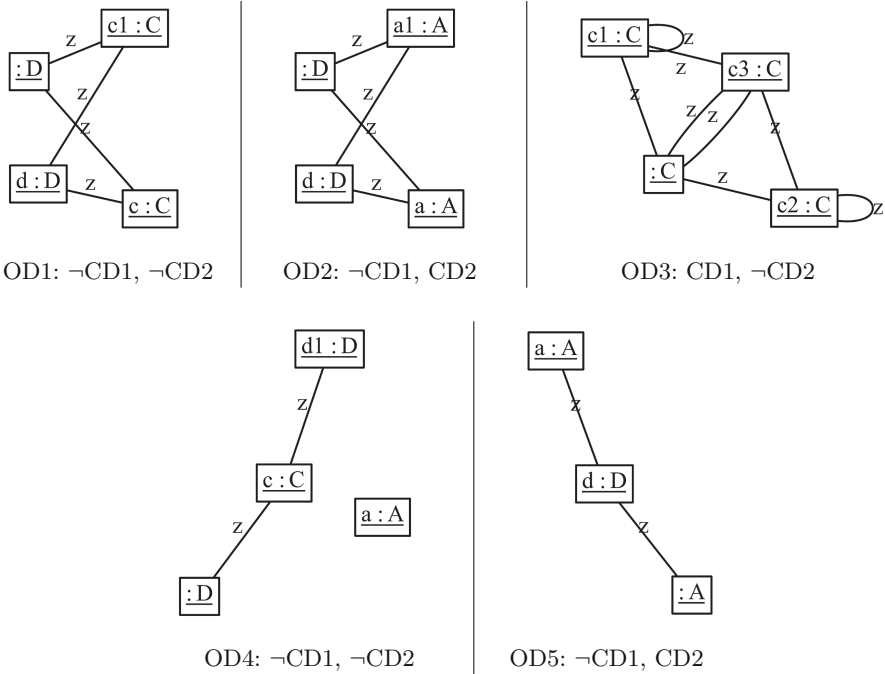


Fig. 11. Randomly chosen ODs in the concrete example run.

absence (showing everything as normal lines instead) to students, just as we do not disclose CD0, CD3, or which mutations have been made between CDs.

7 Related and Future Work

We have already mentioned existing translations from the UML domain to Alloy [1, 7–9, 12] throughout the paper. Instead of generating ODs using Alloy, instances for a given CD may also be found using an instance generating graph grammar [3]. To do so, the meta model in that context would require essentially two extensions. First, analogously to the customisation of CD2Alloy, support for mutations of CDs would be additionally needed. Second, constraints would have to be put in place to ensure that the ODs generated as instances are (or are not) instances of certain mutated variants.

Besides empirically evaluating our own exercise task generation via use with student cohorts, it would be interesting to further investigate ways of tailoring tasks to specific teaching goals. We have already discussed some possibilities, such as in the last paragraph of the previous section. We could also provide even more control to instructors for variability of tasks, for example not just letting them configure the numbers of classes, relationships etc., but also which CD mutations should be employed in a certain setup (thus allowing generation of

tasks focusing on a specific CD/OD concept, or in which we can actually ask students for the reasons a certain OD does not conform to a certain CD), or allowing a larger mutation distance between the CDs used in a task. Extending the approach in order to go beyond structural aspects of CDs and ODs, for example by including attribute fields and methods, would be feasible since the CD2Alloy translation already supports these features. Of course, we would have to use didactic considerations, such as which kinds of bad examples related to attributes and methods we want to handle, for devising appropriate mutations to employ. On a more technical level, future changes could see us using Alloy for generation of CDs (under a range of structural and possibly other constraints) as well. And in a departure from our current use of completely artificial class and object names, we could aim for generating tasks with more meaningful names, also for attributes etc., instead.



References

1. Anastasakis, K., Bordbar, B., Georg, G., Ray, I.: On challenges of model transformation from UML to Alloy. *Softw. Syst. Model.* **9**(1), 69–86 (2010)
2. Booch, G.: *The Unified Modeling Language User Guide*. Pearson Education India (2005)
3. Ehrig, K., Küster, J.M., Taentzer, G.: Generating instance models from meta models. *Softw. Syst. Model.* **8**(4), 479–500 (2009)
4. Fowler, M.: *UML Distilled: A Brief Guide to the Standard Object Modeling Language*. Addison-Wesley Professional, Reading (2004)
5. Jackson, D.: Alloy: a lightweight object modelling notation. *ACM Trans. Softw. Eng. Methodol. (TOSEM)* **11**(2), 256–290 (2002)
6. Jackson, D.: *Software Abstractions – Logic, Language, and Analysis*, Revised edn. MIT Press, Cambridge (2011)
7. Kautz, O., Maoz, S., Ringert, J.O., Rumpe, B.: CD2Alloy: a translation of class diagrams to Alloy. Technical report AIB-2017-06, RWTH Aachen University (2017)
8. Maoz, S., Ringert, J.O., Rumpe, B.: CD2Alloy: class diagrams analysis using Alloy revisited. In: Whittle, J., Clark, T., Kühne, T. (eds.) *MODELS 2011*. LNCS, vol. 6981, pp. 592–607. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-24485-8_44
9. Massoni, T., Gheyi, R., Borba, P.: A UML class diagram analyzer. In: *Proceedings of Workshop on Critical Systems Development with UML*, pp. 143–153 (2004)
10. Object Management Group: *Unified Modeling Language (OMG UML)*, Version 2.5.1, December 2017
11. Rumbaugh, J., Jacobson, I., Booch, G.: *The Unified Modeling Language Reference Manual*. Pearson Higher Education (2004)
12. Shah, S.M.A., Anastasakis, K., Bordbar, B.: From UML to Alloy and back again. In: Ghosh, S. (ed.) *MODELS 2009*. LNCS, vol. 6002, pp. 158–171. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-12261-3_16
13. Waldmann, J.: Generating and grading exercises on algorithms and data structures automatically. In: *Proceedings of Automatische Bewertung von Programmieraufgaben*. CEUR Workshop Proceedings, vol. 2015. CEUR-WS.org (2017)

Education of Post-Graduate Research-Students



A Connectivist View of a Research Methodology Semantic Wiki

Colin Pilkington¹ and Laurette Pretorius²

¹ Department of Computer Science, University of South Africa,
Florida Park, Roodepoort, South Africa

pilkicl@unisa.ac.za

² College of Graduate Studies, University of South Africa,
Muckleneuk, Pretoria, South Africa

laurette@acm.org

Abstract. The use of virtual learning spaces for learning and teaching needs to be underpinned by a pedagogy that provides a basis for the approach used. Connectivism takes a networked view of knowledge; its characteristics and understanding of learning were investigated. Here, the structure and development of a research methodology semantic wiki are described, including how the contents of the wiki allowed for the exploration of the structures of various research methodologies. Positive evaluation of the wiki was obtained from our research students.

Keywords: Connectivism · Semantic wiki · Research methodology · Distance education · Post-graduate students · Online learning

1 Introduction

Virtual learning spaces must be implemented w.r.t. a pedagogy that informs the use of such online approaches [18]. While it has been argued that there is no single learning theory that can be used to understand online learning [30], it has been suggested that current learning environment development is driven by technological advances rather than a considered pedagogy [10]. Thus, the drive to use alternative, often online, approaches to face-to-face modes of delivery of learning and teaching [9] needs to be achieved within a framework that is pedagogically sound. This would be true, too, of the research education that accompanies postgraduate research supervision.

Understanding *research methodology* is fundamental to good research and developing competent researchers [8]. However, the research methodology domain is widely believed to be difficult to learn in that it is both conceptually complex and technical [28], leading to students having difficulty dealing with the diversity of conceptions of the domain, with little consistent understanding of the constructs involved. There is “a lack of shared language describing important foundational concepts of research methodology” [8] (p. 230). Students are

frequently concerned about the difficulties associated with research methodology and typically bring misconceptions about the domain into their studies [14], leading to calls for clearer and more concrete distinctions to be made between the various constructs that make up a research methodology, as well as an understanding of the relationships between them [32]. Blended approaches that extend research education to online tools are being used to support this learning [14, 19].

Advances in technology have led to alternative forms of presenting research methodology education, including web-based approaches [17]. It has also been noted that the growth in participatory technologies and Web 2.0, in which much of current social media is situated, has altered the environment in which interaction is enabled, information is accessed, and knowledge is created, allowing anyone to connect and share with others in the creation (and publishing) of this knowledge [15]. Online environments used for teaching and learning purposes have moved past institutional learning management systems to virtual communities of practice [21], where little is done in isolation, and are characterised by more social and collaborative models of learning [18]. Here, students are immersed in situated networks of social relationships of learning and shared practice with supervisors, other academics, and peers [11, 30]. However, such virtual communities are still in their beginning phases, and the role that Web 2.0 technologies play in these virtual learning communities still needs to be explored further [21].

The *Semantic Web* is one area where knowledge representation and integration with e-learning can have an impact on higher education [29]. Semantic web technologies support linking data using semantics, which help provide meaning to the link, and so supersede the basic linking that Web 2.0 provides [33]. Combining semantic web technologies with learning theory and teaching and learning practice is producing interesting results, although it is still at an early stage of exploration [29]. Interestingly, the Semantic Web is not yet recognised by the NMC Horizon Report [3] as one of the enabling technologies that will transform what can be expected of online tools in higher education.

Recognising the role that the Semantic Web can play in knowledge representation, and the necessity for researchers to master research methodologies, the question explored here is to appreciate to what extent a learning framework provided by connectivism can be used to understand the use of the affordances provided by semantic technologies in the learning of research methodology structure by postgraduate students.

The remainder of this paper is organised as follows. An understanding of *connectivism as a learning model* will be presented next, followed by the description of a semantic wiki employed to explore research methodology structure. The connections between the two, showing how a semantic wiki can be seen as an implementation of connectivist approaches to learning, will be discussed. Finally, some conclusions will be drawn.

2 Related Work

The learning environment has changed in the last 15 years [7], and knowledge is no longer seen as immutable (something that can be learnt once and is known or mastered forever), but is now seen as the ability to find and create knowledge rather than simply consume it [15]. Knowledge and learner management solutions have often failed as a result of the heavy dependence on content and/or technology [6], whereas a connectivist approach leads to a shift away from knowing *what* to knowing *how* or *who* and even *where* [31]. *Connectivism* recognises that the ways in which knowledge flows have changed substantially as a result of the data communication networks that have become available [30].

2.1 Connectivism

There has been a move from more behaviourist and cognitivist theories of learning [7], through constructivist and social constructivist theories, to Siemens' connectivist theory of learning [15,31,35]. It is an approach that is not built on past learning theories [2], although connectivism was influenced by social constructivism and the growth of technologies that allowed online participation and collaboration [15]; it may be characterised as networked social learning [13]. It must be noted that there is some disagreement about whether connectivism is a theory of learning, or a pedagogy and model of learning [2,13]. However, here connectivism will be used as a conceptual framework in which to understand a semantic wiki approach, as it is believed that it is a valuable contribution to the ideas of learning within a technologically connected (and networked) world [13].

Connectivism considers knowledge to have a distributed structure [11]; that is, knowledge can be seen as a network with nodes, with a node being any object that can be connected into a network of some sort [2], and the most effective and reliable way of accessing knowledge is via these networks [12,33]. These nodes can be understood at different levels, from the lowest (the neural network in the brain), to the conceptual or internal (the thoughts and ideas that humans use to interpret the world), to the external (which can be made up of a range of node types and information sources, including people, books, websites, programs, and databases) [1].

These nodes are then linked by interactive relationships, where this link may have direction, may have an inverse link, and may even connect back to the node itself [2]. Concepts then grow by connecting to other concepts [2], where a group of connections seen as a whole is known as a pattern [1] that holds meaning. This pattern may itself be considered a node, so that a node may contain a network of its own, where the node is made up additional nodes [2]. Connectivism holds that such composite, pattern nodes are greater than the sum of their constituent parts [2]. Although knowledge is conceived of as having structure, this structure is not necessarily well organised, is complex, may be chaotic, and does not have layers or a hierarchy; furthermore, the relationships between nodes can be active or inactive [1]. This implies that, as concepts connect to other concepts, the link strength may vary from person to person, leading to different ideas, and meaning, in knowledge networks [2].

The role of technology is emphasised differently by various authors [1], although it has been argued that it can play a role both as actor (such as an artificial intelligence agent on the Semantic Web) and connector (the Internet itself) [2]. Certainly these digital information and communication technologies allow students to follow links in the process of exploring new information [33], whereby the connector allows node relationships to current information to be built more easily [2].

2.2 Connectivist Learning

According to [12] (p. 8), “*knowledge is embedded in the mesh of connections, such that, through interaction with the network, the learner can acquire the knowledge*”. Learning is thus a process of network formation and pattern recognition and acquisition, distributed across a social network of connections [20], and what students can reach in the knowledge network while exploring, and finding patterns, is considered learning [2]. Also, better connections lead to better flow of information [1]. Learning is, therefore, not acquired (and one cannot rely solely on what an individual knows to make good decisions). Rather, knowledge is “*knowledge of the interaction*” [11] (p. 78) between entities, and learning is the ability to access and navigate these knowledge networks, seeing and building connections between concepts and finding and evaluating information [33]—i.e.: learning as “*actionable knowledge*” [31] (p. 4). Connectivist approaches, which focus on connections rather than frequently changing current content, allow thus for rapid changes in both learning context and content [33]. “*The pipe is more important than the content within the pipe*” [31] (p. 5), and knowing where to find updated information is more valuable than remembering its current state [33]. Additionally, in this approach, the student becomes a member of a learning network and is a node, too, that can connect with other students or nodes [1]. This leads to collaborative approaches to learning.

According to [31] connectivism also has implications for the design of learning environments; and instead of a content push design, there needs to be an acknowledgement of the contribution connectivism makes to learning theory—hence the need for new models that reflect this approach to learning and knowledge [6]. Concepts should be seen as forming a network rather than simply being linear [13].

However, connective knowledge is “*no magic pill, no simple route to reliability*” [11] (p. 100). It remains one approach to knowledge that can be used to examine learning and teaching practice. Furthermore, connectivism is not without its critics [1,35]. The argument is that, with the focus on what constitutes learning in a connected world, there is no clear account of how connections are made and how learning is achieved. Additionally, it is not really anything new, and current theories (behaviourism, cognitivism, and constructivism) [7] are sufficient to deal with technology in learning. There are also concerns that it is not testable and that it underplays human interaction.

2.3 Research Methodology Education

A focus on research methodology education is more than simply a matter of providing a postgraduate student with online resources, and it can be anchored in a theory of learning that takes cognisance of the networked, and continuous, nature of learning [15]. This pedagogy can focus on a learning community where, through collective, diverse contributions, connections, and reflection, there is the negotiation of a collective understanding and meaning [15]. With the exploding nature of the access to information, including research articles, it would seem that this model of learning is capable of expressing how postgraduate students gain knowledge about research methodologies. Also, a virtual learning environment should be a tool to help build interconnections between research methodology constructs, allowing the research student an opportunity to make connections between pieces of information and extending these to further maintain and build his/her networked knowledge. Additionally, the Semantic Web and semantic computing tools could conceivably make this networked knowledge machine processable, leading to dynamic knowledge representations and automated reasoning about such representations, with a positive effect on further networking of knowledge and increased learning. Considered w.r.t. the *eight principles of connectivism* [31], learning about research methodologies is centred on the process of connecting research methodology conceptual nodes using appropriate relationships, including the learning that may be found in a Semantic Web environment, and cultivating these connections to ensure continued learning. Not only is the ability to see the connections between the various concepts and relations embedded in research methodologies a core skill, but learning and knowledge in such an environment rely on the variety of views and opinions contained in the domain and the decision-making that is required when choosing which connections to hold on to w.r.t. current knowledge. Ultimately the emphasis is on the capacity to not only know more, but to also have access to accurate and up-to-date knowledge. However, while research methodology courseware can be delivered online at least as successfully as more traditional approaches, with similar student performance [5, 17], there is appreciable variation in experience [25]. Although online participation has been linked to wider opportunities for growth and higher assessment marks, it may not be the preferred approach chosen by some students [5, 25].

3 A Semantic Wiki for Research Methodology

A *semantic wiki* is a merging of the benefits of social software (such as a traditional wiki) with the Semantic Web [27]. It allows for the creation of semantically enriched, formalised domain content that supports collaborative knowledge production and presentation [34]. Web pages are then at least partially machine processable after being tagged with a concept or property name, and queries can also be achieved using the query language of the Semantic Web, SPARQL [16]. A number of semantic wikis were developed after the initial wiki in 2004, with much of the effort happening around 2005 and 2006 [4]. Semantic MediaWiki (SMW),

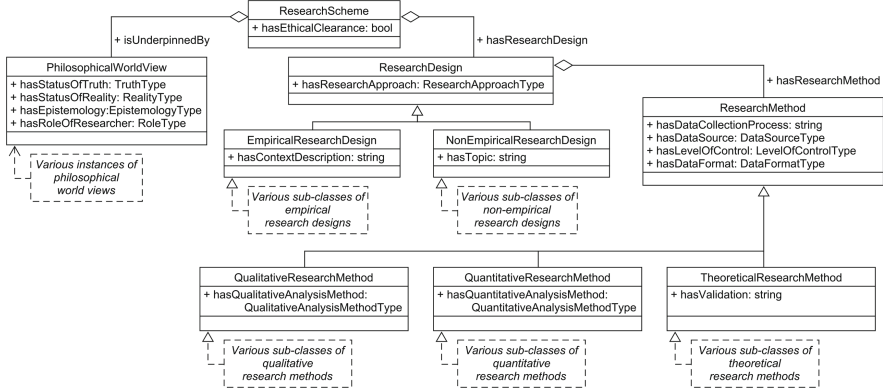


Fig. 1. UML class diagram of our conceptual model

the wiki used for the research methodology wiki, is an open-source extension of MediaWiki [34], which is the engine used to create the well-known Wikipedia, and is considered the most popular semantic wiki engine [34].

3.1 Our Semantic Wiki

Developing our semantic wiki required an *ontology of the domain*, which, in turn, required a conceptual model. An ontology engineering process was applied to develop such a conceptual model of research methodology structure, followed by an ontology built in Protégé, which was then implemented in SMW.¹ In our approach, a *research scheme* is a container for the components that make up a research methodology (Fig. 1). It consists of a ‘philosophical world view’ that underpins the research, a ‘research design’ that provides the structure of the research, and ‘research methods’ that are used in a research design to gather data.

Our main landing page describes the overall structure of a research methodology as well as indicating how the wiki could be used. Other pages describe how to explore the semantics of the wiki, make comparisons between this research methodology structure and others that have been proposed, and indicate how to edit existing pages, add citations, and create new content. A graph view provides functionality to explore research schemes graphically and a link to a special page that allows users to explore any of the categories in the wiki. A ‘breadcrumbs’ feature was added to provide links to the last five pages visited. The text on each page would be the main content of the wiki, with an associated Discussion page allowing the content, and the justifications for or against it, to be separated. This supports collaborative work, as it enables users to present the main ideas concisely, while, at the same time, using an accompanying page to discuss and argue about the rationale for the content.

¹ http://eagle.unisa.ac.za/mediawiki/index.php/Semantic_Web_and_Research_Methodology.

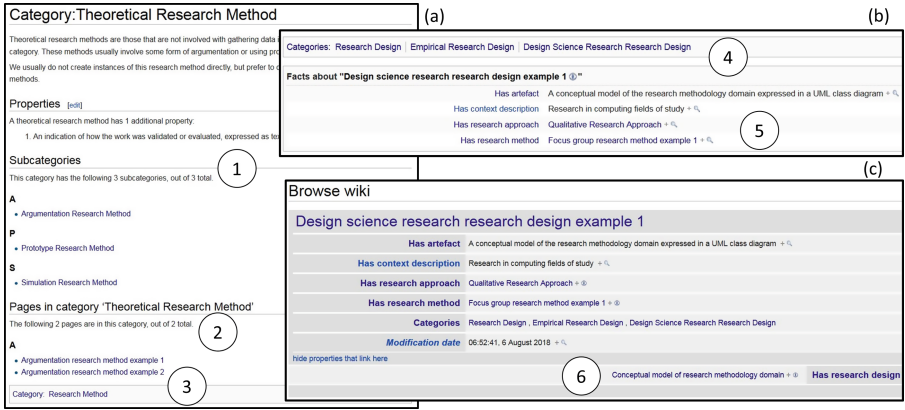


Fig. 2. (a) An example of an SMW *Category* page, (b) An example of an SMW fact box, (c) The SMW Browse wiki view

Categories and Properties. Most pages belong to some ontological entity, having different namespaces to differentiate between, and classify, the entity types [34]. The main and data type classes are represented as *Category* pages, where data type classes are used for entity attributes. All properties (both object properties that point to other entities/objects and data properties that implement entity attributes) are represented as *Property* pages, where the target is the value of property [34]. Thus:

- the *ResearchDesign* class is realised as the *Category:Research Design* page;
- the *ResearchApproachType* attribute of a *ResearchDesign* is realised as a *Category:Research Approach* page;
- the *hasResearchMethod* object property of a *ResearchDesign* is realised as a *Property:Has research method* page; and
- the *hasResearchApproach* data property is realised as a *Property:Has research approach* page.

Category and *Property* pages were populated with basic data describing the entity/property, ensuring that users use them consistently [34]. Even though a *Property:* is represented by a page, it is used to create typed linking from one page to another page or data value. Each individual (or instance) in the ontology is also implemented as a separate, normal page. Thus, the *Pragmatism* individual of a *Category:Philosophical World View* has a page of its own and would contain all the attributes of a *Category:Philosophical World View* as well as a description of the world view. Where the individual is of a class that is lower in the class hierarchy, such as an instance of a case study, it would contain all the attributes of the superclasses, that is, of *Category:Research Design*, *Category:Empirical Research Design*, and *Category:Case Study Research Design*, as well as some extra detail pertaining to that particular case study individual.

Annotations and Browsing the Wiki. Annotations are used to make semantic statements about entities in SMW. Even though individuals, categories, and properties are realised as separate pages, the annotations refer to the concept discussed on the page rather than the actual web page itself [24]. These annotations are added to the wikitext using a simplified markup format [34], making page semantics machine readable. For example, adding an annotation on a normal article page declares that page to be an instance of the specific concept. These annotations are used for the instances (or article pages) of specific research schemes, philosophical world views, research designs, research methods, and data types used as entity attributes. When such an annotation is added to a *Category* page, it declares it to be a sub-class of the given category; these annotations were used to set up the inheritance hierarchy for research designs and research methods.

The result of such annotation is that when a *Category* page is displayed (Fig. 2a), the subcategories of that page (point 1), the article pages of that category type (point 2), and the page category type (point 3) are displayed dynamically. A further advantage of such semantic annotation is that it allows intelligent browsing of the wiki [34]. Semantic information is dynamically displayed at the bottom of each ordinary article page (Fig. 2b): a Categories box indicates what kind of page this is, where the whole category-subcategory hierarchy is shown (point 4). A fact box displays all the annotations on the page in a linked format, allowing a user to click on a property link (on the left) to visit that property's page (and see other individuals where this property is used) or to click on the property's value (on the right) if the value is represented by another article page (point 5). A user can also access an inverse link search by clicking on the eye symbol to the right of the page name in the fact box. This takes the user to the *Browse wiki* page view (Fig. 2c), which shows the links that point to the current page (point 6). It is thus possible to follow this link back to the specific page that points to the current page or to click on the property link that was used to link the two pages. This *Browse wiki* page can be accessed from any link on any page.

Queries. SMW has an easy-to-use inline query engine that allows a query to be included on a page, which then provides updated, dynamic results when the page is accessed. For example, the query (`#ask: [[Has case study design:: SUBJECTPAGENAME]]`) can be used to display all pages that have the property *Has case study design* that point to the current page.

Wiki Individuals. To test the functionality of the wiki and to provide content, research articles were read, manually extracting the research methodology structure used, and added as instances or individuals of *Category:Research Schemes*. The provision of attribute data is not required to allow for cases where reported research might not have mentioned the attributes that have been included in the conceptual model on which the wiki was structured.

3.2 Students' Evaluation

With permission by our university, a link to the wiki was sent to all 316 students registered for an 'honours' research report module,² and they were later sent a link to the questionnaire. This was a non-probabilistic, self-selected survey, and the results may not be representative of the entire research student population. Fifty-nine responses were received, yielding a response rate of 19%.

Demographic and Background Information. The respondents were mainly males in their 30s (40%), followed by females in their 30s (26%). Of these, 98% considered their Internet expertise level as good or expert, with 95% using online communication regularly. In total, 96% indicated a strong enjoyment of online tools, although only 53% used social networks often; 86% had used a wiki 10 or more times; and 78% had never contributed to one.

Using the Wiki. Table 1 summarises our students' responses to the wiki; percentages may not add up to 100% due to some non-responses. Our students found the wiki easy to navigate and indicated that it provided valuable information and helped them to understand research methodology structure. However, 81% of the students did not contribute to the wiki, mostly indicating that they did not have enough knowledge (38%), did not think it was necessary (23%), or had no time (21%). Of those who did contribute, 23% were very confident of their contributions, and 62% were sure about them; 67% found it easy to contribute, while 25% noted that it became easier as they progressed. Overall, 43% enjoyed using the wiki, 72% found it useful, and 50% indicated that it made them think and that they would use it again. Only 36% would recommend it.

Table 1. Student responses to our semantic wiki, on a Likert scale with (5) = strongly agree, (4) = agree, (3) = not sure, (2) = disagree, (1) = strongly disagree

Statement	(5)	(4)	(3)	(2)	(1)
The wiki was easy to navigate	45%	45%	3%	0%	0%
I could understand the research methodology structure	26%	59%	9%	2%	0%
The wiki provided valuable information	41%	55%	3%	0%	0%

Themes. Seven themes were identified in the textual responses given by participants. Many found it *useful*: as the most comprehensive, easy-to-understand structure they had ever seen on this topic. However, some noted that *more was required*, as it would not be enough to ensure effective learning; a 'question and answer' functionality ought to be included. Some students also wanted more of an *overview*, like a high-level road-map, or a stepwise presentation which would be easier to follow. Also more *resources* were asked for, such as links to research

² The South African 'honours' degree is an extension to our 'short' bachelor degree, approximately comparable to the final study-year of the (longer) US-American bachelor degree. It is typically a prerequisite for starting a master project in South Africa.

methodology papers or referencing software. In several cases the instances of research schemes were either not found or too few, which lead to requests for more intuitive *examples* of each methodology. Finally, there might be some *lack of confidence* on the part of students to add content, as they were not always sure whether their own contributions were correct.

4 Discussion

Some students clearly found value in the semantic wiki, and connectivism can be employed as a theoretical framework from which to explore the source of this value of an applied semantic approach to teaching and learning the structure of research methodologies. It has previously been noted that semantic web technologies and ontologies, which can be used to set up the formal specifications of concepts and relationships, are able to operationalise the principles of connectivism [33]. While it has been noted that Wikipedia can be seen as an instance of connectivist knowledge [11], the extent to which the research methodology semantic wiki can be understood to be a valid approach to presenting domain knowledge will be discussed here. It is noteworthy that connectivism has been used before as an argument to support the ongoing learning that occurs in a knowledge-based engineering environment [20].

4.1 Nodal Structure

The conceptual model of the research methodology structure used in the wiki has a definite hierarchical structure with typed links between the four main entities:

1. the *research scheme* as a container for
2. a *philosophical world view* and
3. a *research design*, where each research design contains
4. appropriate *research methods*.

Included in this structure are links to object attributes for the types of designs and methods. This structure fits well with the connectivist concept of knowledge as structured [2]. Furthermore, it supports the idea that knowledge of research methodologies can be conceptualised as a network that is not just a flat, linear set of entities [13], but that the links/connections between the entities carry semantics and meaning. The semantic wiki provides the connections between concepts, providing a pattern to be discovered.

This structure can also be interpreted as *nodes* (Fig. 3; text in italics will refer to the specific detail in this figure). The whole site may be seen as a node to which a knowledge network can link as a place to find information about the structure of a research methodology. One level of granularity lower, a whole research scheme may be seen as a node; these nodes can be taken as instances of research reports that have been published in journals and conference proceedings and are, in a sense, self-contained. The research scheme concept grows by connection to other concepts [2] like ‘ethical clearance’, philosophical

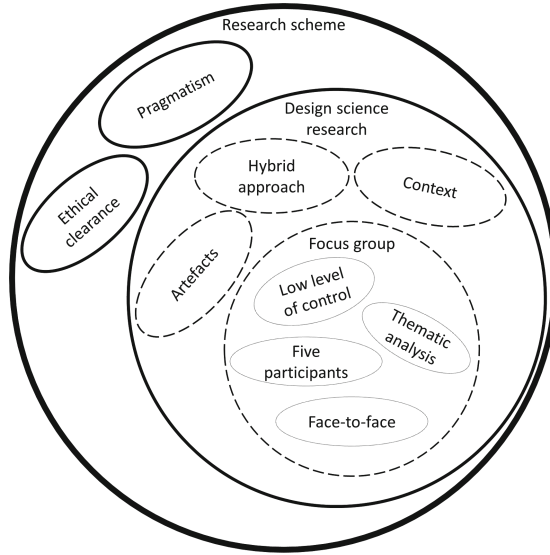


Fig. 3. Zooming into a research scheme node

world views, and research designs. A research scheme can also be interpreted as a ‘pattern’, which is a set of connections tied together as one whole [2], whereby the meaning about the roles of the included parts is encapsulated in this pattern.

Continuing into the structure, it is possible to zoom into one of the research scheme nodes to find the sub-nodes contained within it and to explore how these are structured: it contains a link to a world view (*Pragmatism*), some indication of an ethical clearance, and a link to a research design (*Design science research*). Zoom into the design node to find an approach type (*Hybrid approach*) and other nodes specific to the type of design being used (*Context*, *Artefacts*), as well as a link to a research method (*Focus group*). Zoom into that node to find nodes that give detail about the specific method that was used (*Low level of control*, *Five participants*, *Thematic analysis*, *Face-to-face*). Additionally, following a link from one of the properties (such as the hybrid approach in the design science research node) will take the student to other types of approaches that could have been used.

Any technology-enhanced environment, such as a semantic wiki, that is to support a connectivist approach to learning needs to structure or organise knowledge and handle the connections, so that information is discoverable [33]. A semantic wiki is also able to handle the dynamic nature of growing knowledge through the queries that can be placed on pages. Thus, as new instances (with their associated links) are added to the structure, these will show up automatically on the appropriate pages. This, again, emphasises a connectivist model of knowledge, its changing nature, and the importance of knowing where to find the most current information.

Furthermore, it has been argued that nodes have ‘autonomy’ to such an extent that concepts can *accept or reject connections* to other nodes largely as a result of the connections that are already linked to concepts. This leads to differences of opinion and reasoning [2]. Thus, although the structure (nodes and links) is provided by the semantic wiki, the content of a node, and strength and status of a link (active or inactive), in the mind of the explorer of the wiki are not always the same for all individuals. Hence there will be different ideas about the value of the wiki. Thus, some supervisors or students may find the structure useful—others not.

4.2 Learning

Semantic wikis can be seen to facilitate learning when viewed from a connectivist model of learning. When presented in a semantic wiki, a research student is able to follow typed links, promoting a connectivist approach to learning [33], as the student explores the networked knowledge about research methodologies present in the semantic wiki. A student is able to see the interconnectedness of the concepts by following the links to more information and can so build paths of knowledge through the chaotic maze of terminology that characterises the domain. Furthermore, the connectivist view of learning as pattern recognition applies here: students see, and can acquire, the pattern of linkages and relationships that constitute research methodology concepts, getting the meaning represented by the pattern to be accepted by the current concepts that are held.

Also, when two concepts are connected, they allow knowledge of the one to be transferred to the other [2]. In the semantic wiki, since one research design can be replaced by another (as they are seen to be connected by the inheritance relationship), it is possible to transfer the knowledge that the student has about one research design (i.e.: by what it is constituted and how it relates to other parts of a research methodology) to some new design, although some specifics of it will have to be reorganised.

Connectivist learning has been characterised by *four activities*—namely: ‘aggregate’, ‘relate’, ‘create’, and ‘share’ [20, 23] (also called ‘aggregate’, ‘remix’, ‘repurpose’, and ‘feed forward’ [1])—and *four levels of interaction*, namely: ‘operation’, ‘wayfinding’, ‘sensemaking’, and ‘innovation’ [22, 35]. These may be reinterpreted for explaining learning in this research methodology semantic wiki, as well as for considering the needed critical skills, as follows:

1. *Operation*: initially, students need to master the technical human-wiki interface necessary to participate in the learning available in the wiki. This basic interaction points to a critical literacy required to be an effective connectivist learner using this wiki.
2. *Aggregation and wayfinding*: students access the resource, learn to navigate it, and build connections between nodes that they find reliable within it. In aggregating concepts around a research scheme, for example, students learn what it consists of and how the parts relate to one another. Students also need to judge the content and connections to determine what is important

and valuable, (i.e.: another critical literacy is required). Students orientate themselves in the spacial structure presented by the wiki and develop a loose network.

3. *Relate, remix, and sensemaking*: students reflect on what they have found and use research scheme instances to relate to their own experience and how past research has been conceptualised and patterned. In sensemaking interactions, they construct patterns of meaning and understanding (leading to a consistent comprehension) and remix concepts from different domains (rearranging parts to meet their needs by changing some connections to link to more appropriate nodes or concepts for their particular research). The result is a tighter network. Here, critical analysis skills are needed.
4. *Create, repurpose, and innovation*: students now create something of their own; they build their own research schemes from the knowledge that they have gathered and reworked within the network and so build up their own patterns. Thus, a certain level of ability to create and innovate is another connectivist critical literacy, and innovation interaction is the deepest, most challenging, and applied level to reach.
5. *Share and feed forward*: students then share what was created with others, and the discussion pages in the wiki further allow students to share their ideas about why choices were made and to discuss these with other people.

By actively using the wiki in getting students to comment on the discussion pages about a research scheme or its component parts, supervisors will be supporting students to aggregate. Furthermore, students could use the wiki to construct their own research methodology pathways in the wiki and justify their choices, which would take students through the other three phases of learning via a semantic wiki. Thus, in a sense, students become content generators, as they restructure the information contained in the patterns they have seen in the semantic wiki to form new patterns that they can use in their own research methodology [2]. The semantic wiki is thus able to act as the place of interaction between supervisors and students, which leads to knowledge [11], and further consolidates the link between the wiki and the connectivist learning model. In this study, the extent to which students created research schemes for their research reports is not known. However, there was no sharing of ideas evident in the wiki, as it appears that there is little confidence among the students to engage. This result has been reported before, where only a minority of students created an artefact [23].

Since the new aggregation or organisation of existing knowledge is *new knowledge* (because such compounded nodes are greater than the sum of their parts) [2], students, in gathering the parts of a research scheme (to define a research methodology structure for use in a specific research project with specific questions), are actually learning and generating new knowledge as they work. In some senses the work of a supervisor is to find the best way of using such networked knowledge in order to enhance a student's learning experience [1]. It is necessary, though, to appreciate the level of autonomous, self-directed learning that is called for in connectivist learning when students have to find resources,

make connections, and independently take responsibility for their learning [23]. Furthermore, not the mode of delivery is important, but rather the representation of the content [17]; a semantic wiki may thus be an efficient *tool* for using connectivist ideas to *support* students' learning.

5 Conclusion

Although “*learning research methodology is a multifaceted and intellectually challenging endeavour*” [8] (p. 230), it is a task that students undertaking research have to master to some extent if they are to produce acceptable research results. The move to use technological tools in higher education—especially ‘off-campus’ distance learning—including the Web with its access and collaborative affordances has led to alternative approaches to the research education that accompanies the learning of research methodologies. In this paper we described one such attempt that uses the Semantic Web, in the form of a semantic wiki, to support the learning and teaching of the structure of a research methodology. All-in-all it was well received by our students. However, the use of advanced (including online) technology is not necessarily going to lead to better-quality learning or success [26]. Teaching and learning should not be turning to the unquestioned use of technological advances, but rather to a thoughtful practice of pedagogical principles [17]. Connectivism can provide these pedagogical principles in the case of the semantic wiki explored here and lays a good foundation for understanding how semantic technologies may be of value. Furthermore, semantic wikis equip a course designer with tools that can be used for developing, supporting, and maintaining network formation, which would support connectivist learning [22]. The connectivist approach to learning places a focus on a networked view of knowledge and its acquisition, which is strongly supported by the semantics available in a semantic wiki. Also, it encourages the gathering and reviewing of a wide variety of resources, points of view, and judgements of what is of value, before reaching decisions about the creation of a student’s own opinions and new knowledge. In a sense, connectivism allows one to think in new ways about objects of learning and how they can be presented to students [22].

References




1. Al Dahdoh, A.A.: Jumping from one resource to another: how do students navigate learning networks? *Int. J. Educ. Tech. High. Educ.* **15**(1), 45 (2018)
2. Al Dahdoh, A.A., Osorio, A., Caires, S.: Understanding knowledge network, learning and connectivism. *Int. J. Instruct. Tech. Distance Learn.* **12**(10), 3–21 (2015)
3. Becker, S.A., et al.: NMC Horizon Report: 2018 Higher Higher Education Edition. Technical report, EDUCAUSE, Louisville (2018)
4. Bry, F., Schaffert, S., Vrandečić, D., Weiand, K.: Semantic wikis: approaches, applications, and perspectives. In: Eiter, T., Krennwallner, T. (eds.) *Reasoning Web 2012*. LNCS, vol. 7487, pp. 329–369. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33158-9_9

5. Campbell, M.: Teaching, communities of practice and the police. In: Proceedings of the 31st HERDSA Annual Conference, pp. 106–116 (2008)
6. Chatti, M., Jarke, M., Frosch-Wilke, D.: The future of e-Learning: a shift to knowledge networking and social software. *Int. J. Knowl. Learn.* **3**(4/5), 404–420 (2007)
7. Dakich, E.: Theoretical and epistemological foundations of integrating digital technologies in education in the second half of the 20th century. In: Tatnall, A., Davey, B. (eds.) *Reflections on the History of Computers in Education*. IFIP AICT, vol. 424, pp. 150–163. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-55119-2_10
8. Daniel, B., Kumar, V., Omar, N.: Postgraduate conception of research methodology: implications for learning and teaching. *Int. J. Res. Meth. Educ.* **41**(2), 220–236 (2018)
9. Department of Higher Education and Training: White Paper for Post-School Education and Training. Republic of South Africa (2013)
10. Dillenbourg, P., Schneider, D.K., Synteta, P.: Virtual learning environments. In: Proceedings of the 3rd Hellenic Conference on Information and Communication Technologies in Education, pp. 3–18 (2002)
11. Downes, S.: An introduction to connective knowledge. In: Hug, T. (ed.) *Media, Knowledge & Education — Exploring New Spaces, Relations and Dynamics in Digital Media Ecologies*, pp. 77–102. Innsbruck University Press, Innsbruck (2008)
12. Downes, S.: *Learning Networks in Practice*. Technical report, National Research Council, NRC Publications Archive, Canada (2007)
13. Duke, B., Harper, G., Johnston, M.: Connectivism as a learning theory for the digital age. In: Proceedings of the Higher Education Teaching and Learning Association Orlando Conference (2013)
14. Earley, M.A.: A synthesis of the literature on research methods education. *Teach. High. Educ.* **19**(3), 242–253 (2014)
15. Farkas, M.: Participatory technologies, Pedagogy 2.0 and information literacy. *Libr. Hi Tech* **30**(1), 82–94 (2012)
16. Feigenbaum, L., Prud'hommeaux, E.: SPARQL by Example: a Tutorial (2010). <https://www.cambridgesemantics.com/blog/semantic-university/learn-sparql/sparql-by-example/>
17. Girod, M., Wojcikiewicz, S.: Comparing distance vs. campus-based delivery of research methods courses. *Educ. Res. Q.* **33**(2), 47–61 (2009)
18. Hunt, L., Huijser, H., Sankey, M.: Learning spaces for the digital age: blending space with pedagogy. In: Keppell, M., Souter, K., Riddle, M. (eds.) *Physical and Virtual Learning Spaces in Higher Education*, pp. 182–197. Information Science Reference (2012)
19. Ivankova, N., Plano Clark, V.: Teaching mixed methods research: using a socio-ecological framework as a pedagogical approach for addressing the complexity of the field. *Int. J. Soc. Res. Meth.* **21**(4), 409–424 (2018)
20. Johansson, J., Contero, M., Company, P., Elgh, F.: Supporting connectivism in knowledge-based engineering with graph theory, filtering techniques and model quality assurance. *Adv. Eng. Inform.* **38**, 252–263 (2018)
21. Kirkwood, K., Best, G., McCormack, R., Tout, D.: Student mentors in physical and virtual learning spaces. In: Keppell, M., Souter, K., Riddle, M. (eds.) *Physical and Virtual Learning Spaces in Higher Education*, pp. 278–294. Information Science Reference (2012)
22. Kizito, R.N.: Connectivism in learning activity design: implications for pedagogically-based technology adoption in African higher education contexts. *Int. Rev. Res. Open Distrib. Learn.* **17**(2), 19–39 (2016)

23. Kop, R.: The challenges to connectivist learning on open online networks: learning experiences during a massive open online course. *Int. Rev. Res. Open Distrib. Learn.* **12**(3), 19–38 (2011)
24. Krötzsch, M., Vrandečić, D., Völkel, M.: Semantic MediaWiki. In: Cruz, I., et al. (eds.) *ISWC 2006. LNCS*, vol. 4273, pp. 935–942. Springer, Heidelberg (2006). <https://doi.org/10.1007/11926078-68>
25. Lim, J.H., Dannels, S.A., Watkins, R.: Qualitative investigation of doctoral students' learning experiences in online research methods courses. *Q. Rev. Distance Educ.* **9**(3), 223–236 (2008)
26. Maor, D., Zariski, A.: Is there a fit between pedagogy and technology in online learning? In: *Proceedings of the 12th Annual Teaching and Learning Forum* (2003)
27. Meilender, T., Lieber, J., Palomares, F., Jay, N.: From Web 1.0 to social semantic web: lessons learnt from a migration to a medical semantic wiki. In: Simperl, E., Cimiano, P., Polleres, A., Corcho, O., Presutti, V. (eds.) *ESWC 2012. LNCS*, vol. 7295, pp. 618–632. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-30284-8_48
28. Nind, M., Lewthwaite, S.: Hard to teach: inclusive pedagogy in social science research methods education. *Int. J. Incl. Educ.* **22**(1), 74–88 (2018)
29. Pástor, D., Jiménez, J., Gómez, O.S., Isotani, S.: New perspectives in instructional design using semantic web technologies: a systematic literature review. *Ing. Desarro.* **36**(1), 215–239 (2018)
30. Picciano, A.G.: Theories and frameworks for online education: seeking an integrated model. *Online Learn.* **21**(3), 166–190 (2017)
31. Siemens, G.: Connectivism: a learning theory for the digital age. *Int. J. Instruct. Tech. Distance Learn.* **2**(1), 1 (2005)
32. Thomas, G.: Teaching research methods in the social sciences. *J. Educ. Teach.* **37**(3), 366–368 (2011)
33. Vas, R., Weber, C., Gkoumas, D.: Implementing connectivism by semantic technologies for self-directed learning. *Int. J. Manpow.* **39**(8), 1032–1046 (2018)
34. Vrandečić, D.: *Ontology Evaluation*. Doctoral Dissertation, Karlsruhe Institute of Technology (2010)
35. Wang, Z., Chen, L., Anderson, T.: A framework for interaction and cognitive engagement in connectivist learning contexts. *Int. Rev. Res. Open Distance Learn.* **15**(2), 121–141 (2014)



Cohort Supervision: Towards a Sustainable Model for Distance Learning

Judy van Biljon¹ , Colin Pilkington² , and Ronell van der Merwe² 

¹ Department of Information Systems, University of South Africa,
Florida Park, Roodepoort, South Africa
vbiljja@unisa.ac.za

² Department of Computer Science, University of South Africa,
Florida Park, Roodepoort, South Africa
{pilkic1,vdmerwer}@unisa.ac.za

Abstract. In response to the challenge of increasing supervision capacity while at the same time also improving the supervision experience, we used a design science research approach to guide the design, implementation and evaluation of a cohort supervision model for master's students in computing at an open-distance university. This paper describes the implementation of a cohort programme in 2018, the findings from data collected during a focus group with students and supervisors, students' reflective evaluations at the end of the module, feedback from the supervisors, and our reflective notes. Our main theoretical contribution is the cohort model proposed for developing supervision capacity at master's level. Our practical contribution is a method for a practical supervision model for master's students based on the concepts of co-operative learning and conversational theory.

Keywords: Post-graduate supervision · Group supervision · Cohort supervision · Distance education · Design science

1 Introduction

Massification and marketisation of higher education have resulted in increasing numbers of research candidates with different levels of capabilities entering postgraduate studies nationally and internationally [5]. Universities are under pressure because of the growing number of students doing research and the increased emphasis on completion rates. This is particularly the case also at our University of South Africa, as its approach to open distance learning (ODL) is aimed at “bridging the time, geographical, economic, social, educational and

Supported by the South African Research Chairs Initiative of the Government's Department of Science and Technology, and the National Research Foundation of South Africa (Grant No. 98564).

communication distance between student and institution, student and academics, student and courseware, and student and peers" [27]. In the context of little face-to-face teaching, ODL focuses on greater flexibility and removing barriers, leading to wider access to learning, greater student-centricity and support, as well as a focus on student success. Not only has this openness led to significantly increased student numbers; the realities of the South African society led to the admission of students who vary widely in their readiness for postgraduate study, with those from disadvantaged areas and schools lacking logical writing skills [13]. The increasing student numbers, in combination with students' lack of preparedness for postgraduate studies, put pressure on supervision capacity, because the increase in student numbers has not been met by a corresponding increase in the provision of experienced supervisors [3, 5]. Given the risks and the impact of failed supervision, higher education institutions cannot afford to have novice supervisors follow a trial-and-error approach [22]. The need is not only to increase the number of supervisors but also to provide experiential supervision training. This discrepancy between required and available supervision capacity is the rationale for this paper in which we seek to explore the following

Research question: What are the components of an effective model for cohort supervision in distance learning which increases supervision capacity while providing support and experiential learning to supervisors of different experience levels?

The term *cohort model* refers to a group or unit set up as a structure in a community of learning to support intellectual development and knowledge production in postgraduate education research [11]. The use of cohort supervision to address the problem of improving supervision capacity is not new: several previous studies investigated doctoral cohort supervision [10]. Dysthe, Samara, and Westrheim proposed a three-pronged approach in master's supervision combining supervision groups, student colloquia and individual supervision [6]. More recent studies published on master's cohort supervision include [15, 21]. Cohort studies focusing on both master's and doctoral supervision include [23], although not in an ODL context. Manyike investigated supervisor challenges in the supervision of master's and doctoral students in an open distance e-learning institution in South Africa [14]. She suggested collaboration between experienced and novice supervisors as means of enhancing the quality of feedback and communication but did not propose a new model. Similarly, [3] described supervisor development as part of cohort supervision in ODL and proposed a cohort supervision model at 'honours' level.¹ Besides considering master's cohort supervision at an ODL institution, the contribution of this paper lies in the concomitant development of supervision capacity at different experience levels.

¹ For readers from outside South Africa: The 'honours' level in South Africa is a voluntary 'top-up' to a shorter Bachelor degree—somewhat comparable to the final stage of the longer Bachelor curriculum in the USA. The 'honours' level is typically a prerequisite for commencing master-studies in South Africa.

2 Related Work

We used the pattern ‘*postgraduate supervision*’ AND (‘*group supervision*’ OR ‘*cohort supervision*’) to search the ACM, Springer, ERIC, Scopus and Web of Science databases for the years 2013–2018 (in January 2019). The total number of full-papers from each database were as follows: ACM 3484, ERIC 115, Scopus 780, Springer 3526, and Web of Science 457. Thus the number of papers in most of the databases was too large for complete analysis. Therefore we considered only the top 20 papers (rated by relevance) of each database, i.e.: 100 papers altogether—actually 96 after some duplicates were removed. We then read their abstracts and removed 55 papers that did not relate to *post-graduate supervision in our field* (ICT). The remaining 41 papers came from various countries—including South Africa, Australia, the United Kingdom, China, the USA, the Netherlands, New Zealand, Denmark, Finland, Sweden, Japan, Israel, Colombo, Malaysia, and Mauritius—which indicates that our problem is globally relevant. The research methods used in those studies included individual interviews (14) and group interviews (3), surveys (24), focus groups (5), case studies (2), and observations (2).² These salient publications can be summarised w.r.t. advantages, disadvantages, best practices (including cohort models proposed), and critical success factors as described below.

Cohort supervision is proposed as an alternative pedagogy for the supervision of large groups of master’s students in an efficient and effective manner, maintaining quality research and graduate output [15]: this approach improved students’ motivation through peer-sharing of experiences and feedback, as well as students taking responsibility for their own academic progress. Ahern (et al.) noticed enhanced morale, the benchmarking of learning and learning from others’ mistakes, as well as collegiality [1]. The benefit to supervisors included the sharing of ideas, what constituted best practice, and strategies for improving supervision. Addressing capacity constraints is an important motivation for using cohort supervision [1,2], though Choy (et al.) warned that universities need to invest both time and resources for cohort development if such a cohort approach is to yield good results as a supervision model [4].

Considering the *challenges supervisors experienced* in group supervision as part of a guidance and counselling master’s programme, Wichmann-Hansen (et al.) identified three major challenges experienced by the cohort supervisors: (1) promoting equal participation within student groups that are often heterogeneous, (2) ensuring a balance between providing answers and involving students, and (3) recognising and growing students’ analytical skills [28]. Meng (et al.) found that informational contextual factors promoted intrinsic motivation, whereas controlling contextual factors have negative effects [16].

Papers considering *best practices for cohort supervision* emphasise the importance of providing a holistic, integrated approach. For example, [24] mentions the fundamental principles of connectedness, wholeness and being. Hutchings main-

² Those methods are not mutually exclusive; one research design might use more than one of those methods.

tains that group supervision can foster sustained mutual support [9] and proposes technology-mediated interactions which are not tied to a specific location—thus facilitating participation and reducing isolation. Maor and Currie support their argument for the use of technology by focusing on the transformational role that it can play in the move from more traditional, dyadic forms of supervision to more collaborative group processes [12]. Other papers specify the specific elements of best practice, like scaffolding, guiding students in the completion of key learning tasks involved in writing a dissertation proposal independently [21], or using proactive communication to engage in meaningful preparation before meetings [20]. Han and Schuurmans-Stekhoven recommend comprehensive research literacy training [7], which should include the critical search for information, understanding, interpreting and evaluating as well as finally synthesising it.

Various studies proposed *alternative cohort supervision* models. Choy (et al.) investigated the development of postgraduate research degrees cohorts [4]. Their approach included four provisions, namely an initial residentially based workshop, developing a learning community, cultivating scholarship, and spaces for continuing learning. Their interventions resulted in the development of a learning environment that supported students and a culture that was nurturing [4]. Marnewick and Nel proposed an efficient and effective master's programme that would lead to good quality research and improved graduate output [15]. Their findings indicate that peer feedback, sharing experiences in the group, and students taking responsibility for their own progress led to improved student motivation. The benefits of the cohort model to the supervisors included sharing of ideas and best practices, as well as shared strategies for improving supervision [15]. In some of our own research group's earlier work [3] we proposed a pyramid cohort supervision model (PCSM) for supervising computing honours students in an ODL environment. That model was based on cooperative learning, conversational theory and scaffolding, whereby the model purposefully integrated technology as part of student support and collaboration [3].

The critical *success factors* relating to the supervisors include the following: a supervisor's own knowledge [26], availability and willingness to help [29], work load and the pressures of the academic environment [15, 22], and the quality of feedback given in formal supervision meetings which require advance preparation [20]. The importance of feedback is also emphasised in [18] whereby, moreover, the harmony between (co)supervisors has an effect on the supervisor-supervisee relationship. Other skills include coaching, scaffolding and support in articulation and reflection practices [19]. Spiller (et al.) recognised further factors that may influence the success of cohort supervision: cross-cultural environments, co-publishing with students, supervisor and student negotiations to ensure common understanding about important aspects, and written feedback on students' drafts [25]. Njie (et al.) argued that supervisors need to be involved in group activities to counter-act unwanted practices such as 'free riding' (not contributing to group activities) [17]. The number and diversity of the personal and contextual factors affecting cohort supervision signifies the complexity of the task and the expectation of the skills required.

As far as the *students* are concerned, Marnewick and Nel mention cross-cultural issues, barriers related to language differences, the lack of academic resources, unrealistic expectations by students, lack of academic scholarship in students, and the academic pressure experienced by supervisors, as important factors [15]. Also highlighted are barriers to communication with lecturers [21], which is exacerbated by students' level of language proficiency [22]. Furthermore, students' misunderstanding of the scope of postgraduate studies and their lack of critical thinking skills need to be considered [22]. Manyike investigated ODL supervision and identified weaknesses in the following areas [14]: allocating postgraduate students to supervisors without consultation, meeting the needs of students who come to postgraduate studies underprepared by guiding them during the thesis-writing process, as well as the challenges inherent in an ODL model which relies primarily on written communication. Co-supervision as part of cohort supervision was highlighted as more than just a 'safety net' for institutions [18]; it leads to a complex web of both interpersonal and institutional relationships (which carry power) whilst also providing opportunities (as there are many ways in which co-supervision can be organised). Therefore it minimises the risk of dual relationships and increased supervisors' opportunity to experience both leading and participating in groups [4].

In summary, the papers recapitulated above support the argument that cohort supervision has potential for increasing supervision capacity and quality, and that the benefits extend to both students and supervisors. However, research also provides evidence of numerous and diverse problems relating to supervisors, students and their interaction. Complexity leads to the development and implementation of various context-specific cohort supervision models. To our best knowledge (to date), the only approach that specifically addresses the issue of supporting novice supervisors while developing supervision capacity in ODL is the *pyramid cohort supervision* approach for supervising computing honours students [3]. Therefore, we used this model as our point of departure in the research design discussed in the following section.

3 Research Design

For this paper we applied the well-known *design science* method [8]. Our research was guided by a pragmatic philosophy with a *single-case study* [30], whereby the units of analysis are the students. The supervisors (as collectives with the students) are our data collection sources.³ A focus group and reflective questionnaires were used as methods to gather data. The design of the intervention is based on the above-mentioned pyramid cohort supervision model [3], where a design science approach was used together with principles of constructivist learning as an active, social, meaning-making process based on individual and shared experiences [25]. Cooperative learning was also involved in assuming a positive interdependence between students in the cohort while maintaining their individual accountability [13].

³ Permission was obtained from the relevant authorities at our university.

Table 1. Events and actions undertaken in the cohort supervision process

Stage	Event	Actions
1.	Introduction and orientation	Providing a tutorial letter detailing the purpose of the proposal module, tasks, deadlines, resources and organizational support. Providing online resources (including literature) in a wiki
	1st meeting: (March)	Group meeting between students, supervisors, administrative support staff, and practitioners, for feedback on initial research questions
2.	Research questions/design	Individual meetings with supervisors, informal group interaction
	2nd meeting: (May)	Presentations to group and external supervisor on literature review, research questions, research design
3.	3rd meeting: (August)	Proposal presentations and focus group to evaluate the students' research approaches. (Our reflective questionnaire was distributed only in December, after all assessment marks had been finalised)

For this paper, however, our approach is different in the following ways. We now apply the model to the *proposal development phase* for master's students in computing, thus hoping to use collaborative peer approaches to encourage the students to critique each other's work, and—in so doing—learn how to critique their own work, too. Thus we hope to yield more solid proposals. As far as support is concerned, we enlisted the help of a part-time administrator for organising the interactions and the reporting, and we also involved external domain experts and experienced supervisors as far as necessary (and available). In our practice we introduced face-to-face meetings for student presentations and feedback.⁴

The specific interventions are listed in Table 1. Our cohort consisted of seven students (master's students who registered for the 'research proposal' module in 2018 with a senior supervisor) and three supervisors (of varying levels of supervision experience). The central idea was to support the postgraduate students in the preparation of their proposals by bringing together a cohort of students who would be working in similar fields, so that they can learn from each other.⁵ A *project site* was created on the web-based learning management system that included (amongst others) tutorial letters, background information to the proposal module, as well as links to important resources. Some initial training was offered in the form of a workshop as well as in providing an opportunity for the students to discuss their research topics and questions. This discussion took

⁴ In 'pure' distance education without seminar rooms, technical means like 'Skype' can be used to facilitate 'virtual meetings' via the Internet.

⁵ For comparison see the 'post-graduate school' models in various implementations in different countries.

place in small groups of students as they considered each other's work, as well as between individual students and one of the supervisors.

A subsequent group event provided students with the opportunity to present their work, to develop skills in condensing their ideas into presentation format (introduction, research questions, brief literature review, and proposed methodology) and to present them in spoken words. The students were also expected to critique another student's work and to give constructive feedback, thereby commenting on strengths, weaknesses, and gaps in the other student's argument. Meta feedback was provided by peers, supervisors, and external supervisors who were brought in to add objectivity and new perspectives; this was achieved in a large group with all participants present. A third group meeting was held in which the students again had to present an outline of their proposal to the group and to receive feedback from supervisors and peers. This again took place as a whole-group event.

In the time between these group events, the students submitted drafts of their work to the supervisors for feedback. Students were initially allocated two supervisors, which was however *not* done in a 'classical' primary/co-supervisor arrangement: instead, the senior supervisor in the supervisory group was involved in all participating students' work.

4 Results

4.1 Evaluation Based on the Students' Responses

Reflective Survey. The students were asked to complete a reflective questionnaire focused on their experience of the cohort supervision; our reflective questions are provided in Appendix A. The following discussion is a summary of the insights gained from a *thematic analysis* of the responses. Our findings are structured w.r.t. *benefits* and *drawbacks* of cohort supervision as experienced by the students in the group, as well as the *critical success factors* (requirements to make the approach useful) and recommendations towards improving the model. The different respondents are represented by capital letters in square brackets.

Considering *benefits*, all students noted benefits associated with the cohort supervision process. It provided an overview, allowing students *"to know where everyone is in their studies and not to miss deadlines! It kept me on track"* [F]. The collective nature was noted in the team work *"from both colleagues and the supervisor"* [B], which afforded the students the possibility of tapping into *"collective intelligence for problem solving"* [A]. Another student noted: *"Comments from various persons helped me in writing the proposal"* [E]. The same student also obtained advice *"on what needed to be improved"* [E], whereby awareness was raised *"on some aspects of the proposal that were not clear"* [E]. Furthermore, the approach helped to *"build my confidence by knowing we are all learning and there are no stupid questions"* [B]. The requirement for the students to give presentations *"was very useful"* [D] and *"improved communication and presentation skills"* [C]. Also, the students learnt *"indirectly from other students because you see how they do things during their presentations"* [D].

Considering the *group interactions*, the cohort led to a sense of “*belonging and knowing that you are not alone*” [A]. Additionally, “*our group has a WhatsApp group, and that is great*” [F], (i.e.: peer-group initiatives to enhance communication). Students gained from others in the group. “*They helped a lot by sharing the articles and research papers which they thought they can benefit my research*” [C]. They also contributing to the group: “*I helped them on technical issues such as using referencing tools and explaining what is expected in each section of the proposal*” [A].

The extent to which the cohort process affected the *quality of the work produced* varied. It had its value in highlighting “*what is needed in the research, starting from research topic, problem, methodologies, and literature reviews*” [C]. However, “*we could still be possibly stuck in our silo mentality when approaching our work*”, as “*there was not much robust discussion on the WhatsApp group*” [A]. Student [F] responded: “*Not really*”. Students also noted drawbacks from participating in cohort supervision. One student mentioned “*limited time given to a student*” [A] to do a presentation due to the size of the group, and suggested workshops to be held more frequently. Too little contact was noted by another student: “*the workshops are too little*”; it would be better to “*perhaps have one every second month*” [G]. One student had not learned anything from the other students, while the other six students stated that they had learnt much from fellow students. This may be because the particular one student had started later in the year and also had a distinctly different project topic.

Success factors often focussed on contact opportunities: “*Regular meetings encourage members to engage and exchange knowledge, give sufficient time to each student within a group*” [A]. Also [G] wrote: “*Regular contact with students*”. The role of the supervisor was also mentioned: “*Commitment. The lecturer was there for us and responded to emails on time*” [B]. Student [E] mentioned: “*Evaluation, comment or feedback from all the supervisors from that group*”. However, the cohort approach by itself was not deemed sufficient: “*one-on-one sessions with my supervisors are an absolute must. That is where I grew and learnt the most*” [F]. “*I would like a mixed approach, as group- and one-on-one sessions are all important*” [A].

Focus Group. A focus group discussion was held with our students to gauge their collective view of the approach that had been taken to their supervision. Five main themes emerged from a thematic analysis of the focus group transcription.

Students commented on their *initial expectations* of the supervision process. Apart from not being sure how it works, they had expected supervision to be based on emails and supervisor meetings. There was the expectation of meeting the supervisor maybe once or twice a month, initiated by both the student and supervisor, “*because if supervisors do not do that, students can sit back and discussions between supervisor and student end up not happening*”. Largely, the students’ expectation was that they would communicate with their supervisors via email, and that supervisors would send out messages via the university’s

LMS, whereby *“if you miss something it is your fault”*. One-on-one meetings with supervisors were liked *“because you get instant feedback and follow-on questions”*.

As far as the *group approach* is concerned, the students were *“comfortable with the environment created and the support given”*, whereby they *“learnt a lot from the workshops”* and also *“from each other”*. Thereby the students also had to learn *“not to react negatively to criticism”*. Knowing what had been covered in the group meetings encouraged one student to realise that *“I am not a quitter; I will try to make success out of this”*. However, it was clear that *“it is not possible for me to take leave days frequently”*, and that group meetings should be *“after working hours, it will give us an opportunity to attend”*.⁶

It is noteworthy that the students set up a **WhatsApp group** by and for themselves. As *“everyone has a phone on-the-go, it is convenient”*, and *“if someone has a question, it is asked and anyone can answer”*. This tool *“created unity amongst the students”*, as well as it built *“a sense of comfort”*, because otherwise *“this journey can be a lonely journey”*. There was an appreciation of the group and a belief that the students benefited from belonging to it. Nonetheless, the students *“have not really shared each other’s papers”*, and did also not appear to *“share and ask the difficult questions”*.

As far as the *future of the group* is concerned, there was a feeling that they enjoyed this method, that it would be good to continue working in this manner, and that the group should not be split along any topic area lines. The group also expressed the belief that its members would be able to advise future proposal students about what is expected in this module.

As the students were expected to present their research topics, *presentation skills* was identified as yet another theme in the focus group discussion. The students took their presentations *“very seriously”*, and prepared themselves by *“watching YouTube videos”*, by checking *“the dos and don’ts and expectations”*, as well as by asking *“experienced friends”*.

4.2 Evaluation Based on the Supervisors’ Responses

As part of the reflective process, the supervisors completed a questionnaire about their experiences of cohort supervision; the questions are listed in Appendix B.

W.r.t. the question of whether the cohort supervision approach met the expectations of the individual supervisors, the respondents reflected that the cohort supervision model provides a platform for *“quality assurance on many levels, including supervision practices, disciplinary content and general research knowledge”*. One respondent indicated that the cohort process addressed some of the anxieties experienced by novice supervisors. This was also emphasised by another respondent who indicated that the process provides *“a safety net against individual biases, inexperience and ignorance for both students and supervisors”* on various levels, including the management of individual experiences, personality clashes, and overall administration problems. Reflecting on the organisational processes, all respondents mentioned that they did not realise the extent of the

⁶ Many of our students are employed in day-time jobs.

complexity of the organisational overheads, and that it would not have been possible to do this without the administrative assistant. One respondent mentioned that, because the projects were all different, it made reading and conceptualising the different projects difficult. The more senior supervisor also commented on the mentor-mentee process and the danger that lies in this process of supervising not only the students but also the less experienced supervisors. One of the less experienced respondents mentioned that more defined rules for both the supervisors and the students should have been set before the project.

The supervisors agreed that the students benefited from the cohort supervision as it assisted the students with peer support on “*emotional, cognitive and organisational*” levels. From conversations with the students it was learned that the students created their own support group, separate from the official cohort group. One respondent mentioned the positive input that has been received from several sources: the advice provided by the external expert supervisors during the initial group sessions, the positive feedback students who attended a post-graduate workshop at a local conference received, as well as the assistance of, and advice from, a post-doctoral fellow. Moreover, the respondents agreed that the approach positively affected the completion rate for the students. One respondent mentioned that, compared to previous years, the students received more input and that the proposals were of a better quality as a result.

In reflecting on what the supervisors would change, the respondents identified that the field of research should be better defined, and that students should be linked to a specific supervisor earlier in the process. This would eliminate the problem of a student redundantly contacting multiple supervisors. The “*lack of a shared platform*” also resulted in too many emails being sent. One respondent commented on teaching the students the skills required to critique their peers’ work. Another one suggested the introduction of a structured presentation template that would support students in presenting (and thus getting feedback on) the critical details of their research design, rather than dwelling on interesting but irrelevant details regarding the rationale for their projects.

5 Discussion and Recommendations

5.1 Discussion of the Results

It is possible to understand the results w.r.t. *shared experience*, a concept that emerged from our literature review. This shared experience added value for both students and supervisors.

The shared journey was for our students an opportunity to see that they were not alone and that they belonged to a group that, together, learnt what was required in a research proposal. The WhatsApp group that they created points to their initiative in supporting each other, if only from a social and administrative standpoint. This built confidence. Another insight was the importance of presentation opportunities were students could get instant and balanced feedback, and could learn together how to present their work in the most efficient manner.

However, the students will not necessarily have learnt from others in all situations, and the need for regular as well as one-to-one meetings was highlighted. It could be argued that a mixed approach that merges cohort and individual supervision is most likely to meet most academic and social needs.

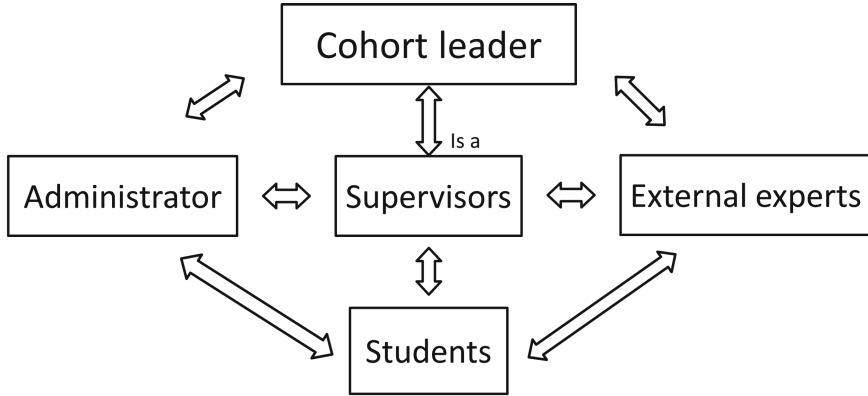
A shared approach which included *external expertise* also had benefits for the supervisors—albeit more for the junior supervisors with less experience than the more experienced ones. This support was noticeable in the ‘backup’ that colleagues offered to early-career supervisors, and in ensuring that quality did not rest on an individual’s shoulders alone. The shared experience entailed a shared responsibility.

However, two further points must be noted. Firstly, a cohort supervision model does *not save supervision time*, and in fact increases the work that a supervisor has to do (as each piece of work is now read by two supervisors instead of one). It is thus unlikely to solve immediate supervision capacity problems. However, this extra effort may be considered as an ‘investment’ in future supervision capacity if we consider the experience that is gained by junior supervisors in the process which may lead to better sole supervision later. Secondly, the expectation that the students analyse each other’s work, and provide constructive feedback which helps them to critically analyse their own work, was somewhat too naive. The comments provided tended to be superficially positive, such that it will be necessary to focus on training students in what to look for when reviewing and analysing academic work in future iterations of our approach. Our findings confirm the value of student colloquia for personal support. They serve as a first filter for ideas and texts, and also bring to light any needs for individual supervision sessions for more specific advice [6]. In our investigation, the supervision groups only started to form towards the end of the first year; perhaps smaller groups would form a better forum for the critical multi-voiced feedback that we found lacking.

5.2 Proposed Cohort Supervision Model

The proposed components of our model for cohort supervision, which is based on the experience described above, are shown in Fig. 1. The components consist of the actors and the relationships between them as well as the recommended resources. The actors include the cohort leader, supervisors, the administrator, external experts, and the students. A cohort leader must be an experienced supervisor who can lead a project and can provide ‘vision’ and guidance where necessary. This person may or may not serve as immediate supervisor to the students. The supervisors are supported by two further role-players: an administrator, who assists in managing the flow of documents and in organising the cohort meetings, and the external experts, who join the cohort on suitable occasions to provide domain knowledge, expert advice, and alternative viewpoints. The students thus benefit from a well-managed process, appropriate supervision, and expert input.

Several resources may be utilised. *Shared resources* are used to benefit the whole cohort and the process of cohort supervision. *Academic resources* are



Shared resources	Academic resources	Cooperative resources	Evaluation resources
<ul style="list-style-type: none"> • Domain knowledge • Experience • Administrative support 	<ul style="list-style-type: none"> • Research literature • Theoretical lenses • Research design • Presentation skills • Academic writing skills 	<ul style="list-style-type: none"> • Workshops • Meetings • Presentations • Digital resources 	<ul style="list-style-type: none"> • Formative evaluations • Summative evaluations • Verbal feedback • Formal written feedback/rubrics

Fig. 1. Proposed cohort supervision model

those that support proposal content development (reading/writing). *Cooperative resources* are ‘places’ where all role players in the process get together to further the supervision process. *Evaluation resources* encompass the processes w.r.t. giving feedback.

5.3 Recommendations

Our findings confirm the benefits of cohort supervision in developing capacity and providing emotional, intellectual and practical support for students and supervisors. Our most important contribution lies in uncovering new challenges related to cohort supervision and in suggesting recommendations to address some of these problems.

Institutional. Institutional practices need to support co-supervision explicitly by providing administrative support, since cohort supervision creates an organisational overhead. Such a support person shall manage the flow of documents, organise cohort meetings, handle queries around registration, bursary applications, ethical research clearance permissions, and the like.

Furthermore, the interactions between the cohort supervisors and the students need to be managed for sustainability. The load on the cohort supervisor can become insurmountable if the cohort supervisor tries to be involved with every student as well as with mentoring the cohort supervisors. If a cohort

supervisor takes on a mentoring role with responsibility for students without being a co-supervisor, then such a cohort supervisor should be recognised as a (meta) supervisor of supervisors in order to facilitate satisfactory progress overall. Institutions need to consider introducing such a role formally in the supervision process. Currently, most higher education institutions have a postgraduate supervision model of sharing credits equally between supervisors and awarding credits per registered student supervised. Mentoring novice supervisors is purported to be important; however if mentoring is not part of the institutional rewards system, experienced supervisors may shy away from the considerable effort and responsibility the role entails.

Structural. The cohort supervision model should clearly delineate the responsibilities to preserve supervision capacity. For example, when two supervisors read the same document for providing feedback, some agreement as to whether this will be done in parallel or sequentially needs to be in place. Also the relative roles of primary and co-supervisors must be defined. Furthermore, while external experts could play a bigger part in helping students to formulate their final research questions and design decisions, how this is to be achieved needs to be negotiated with both students and supervisors, because this process may require extra time to be carried out with integrity.

Organisational. The initial face-to-face meeting, where students can get to know each other and form trust networks with their own social connections, is critical to establishing an informal social support network. Additionally, such meetings provide opportunities for students to do presentations—although the possibility of ‘virtual meetings’ (online) should be further explored.

Academic. The students’ research topics should have sufficiently large overlaps in their ‘theoretical lenses’ and research methodologies. This promotes peer support since the participating students are thus more familiar with the domain and hence better able to constructively critique each other’s work. External experts (and even co-supervisors) can fill gaps when students venture into areas outside the core competency areas of the cohort supervisors, but diverse topics have an efficiency trade-off. The external experts also have a quality assurance role at the proposal acceptance stage.

6 Conclusion and Outlook to Future Work

This paper describes the use of cohort supervision as a way to improve supervision capacity while supporting both students’ research-learning and novice supervisors. Specifically, our findings concerning the implementation of a cohort supervision programme for master’s student at an ODL institution highlight the benefits of our approach for students on emotional, social, cognitive, organisational and quality assurance levels. Nonetheless, institutional buy-in and administrative support will be needed to enable the sustainability of our cohort model. Besides the components proposed for an effective cohort supervision model that incorporates co-operative learning, conversational theory and scaffolding, our

paper also contributes a method for implementing cohort supervision on master's level in an ODL context. As many students at residential universities face time-access- and isolation-constraints, too, our model should be applicable beyond ODL institutions as well. According to the design science approach our proposed supervision cohort model will be applied, evaluated, and reflected on in future research. Future work should consider a longitudinal study to investigate the sustainability of our approach for growing research capacity while also providing satisfactory supervision at the same time. In particular, structural interaction innovations towards improving supervision capacity deserve more attention. Additionally, whereas a qualitative study was conducted here, a more quantitative evaluation (considering, for example, student's pass rates, marks obtained, publications yielded from students' projects, and the like) may provide results that may lead to further refinement of the components of our cohort model.

Acknowledgements. Thanks to *Sewisha Lehong* and *Donald Mothisi* for advice and assistance in analysing our data. Thanks also to the audience at SACLA'2019 for interesting remarks relating to the incorporation of external sources to broaden and strengthen our cohort supervision model.

A Students' Reflective Questionnaire

Supervision for your studies has taken place in a group setting with other students who are on the same journey. Please think about how this process has played itself out and how it has influenced your postgraduate studies, then answer the questions below giving as much detail as you can and are comfortable giving. Please note that you should feel free to be completely honest when answering these questions and none of your answers will determine your research progress in any way. Remember that these questions have no right or wrong answers.

1. Name / Gender / Age
2. When did you first register for your postgraduate studies?
3. What is the current status of your postgraduate studies?
4. Have your postgraduate studies this year been a positive or negative experience for you? Why do you say so?
5. To what extent has the group approach influenced your experience?
6. What has worked, or not worked, for you in this group process? What have been the benefits and drawbacks?
7. To what extent have the other students in the group helped you?
8. To what extent have you helped other students in the group?
9. To what extent do you believe this approach has affected the quality of your work?
10. What do you think are the critical factors for success with group supervision approaches?
11. What would you change about the group approach to supervision used?
12. This was a formal approach to group supervision where you were expected to attend and participate. How would you feel about a more informal peer support approach based on social media (or some other approach)? Would it be more appealing?
13. Would you prefer to continue in this mode of supervision or not, and why?

B Supervisors' Reflective Questionnaire

1. What has been your experience of a group supervision approach?
 - (a) Has it been good or bad?
 - (b) To what extent is it what you expected?
2. From your observations, have the students benefited from the experience or not?
3. Identify challenges and risks in the use of such an approach.
4. How has this approach affected the quality of work submitted?
5. How has this approach affected the completion rates of students?
6. What would you change?




References

1. Ahern, C.M., van de Mortel, T.F., Silberberg, P.L., Barling, J.A., Pit, S.W.: Vertically integrated shared learning models in general practice: a qualitative study. *BMC Fam. Pract.* **14**(144), 1–11 (2013)
2. Anderson, M., et al.: The construction of a postgraduate student and supervisor support framework: using stakeholder voices to promote effective postgraduate teaching and learning practice. *J. Univ. Teach. Learn. Pract.* **15**(2), 6 (2018)
3. van Biljon, J.A., van Dyk, T., Naidoo, L.: Towards increasing supervision capacity: the pyramid cohort supervision model. In: *Proceedings of SACLA 2014 Annual Conference of the Southern African Computer Lecturers' Association*, pp. 166–173 (2014)
4. Choy, S., Delahaye, B.L., Siggers, B.: Developing learning cohorts for postgraduate research degrees. *Aust. Educ. Res.* **42**(1), 19–34 (2015)
5. Cloete, N., Mouton, J., Sheppard, C.: *Doctoral Education in South Africa: Policy, Discourse and Data*. Somerset West: African Minds (2015)
6. Dysthe, O., Samara, A., Westrheim, K.: Multivoiced supervision of master's students: a case study of alternative supervision practices in higher education. *Stud. High. Educ.* **31**(3), 299–318 (2006)
7. Han, J., Schuurmans-Stekhoven, J.: Enhancement of higher degree candidates' research literacy: a pilot study of international students. *Asia Pac. Educ. Res.* **26**(1/2), 31–41 (2017)
8. Hevner, A.R., March, S.T., Park, J., Ram, S.: Design science in information systems research. *MIS Q.* **28**(1), 75–105 (2004)
9. Hutchings, M.: Improving doctoral support through group supervision: analysing face-to-face and technology-mediated strategies for nurturing and sustaining scholarship. *Stud. High. Educ.* **42**(3), 533–550 (2017)
10. Kobayashi, S., Grout, B.W., Rump, C.Ø.: Opportunities to learn scientific thinking in joint doctoral supervision. *Innov. Educ. Teach. Int.* **52**(1), 41–51 (2015)
11. de Lange, N., Pillay, G., Chikoko, V.: Doctoral learning: a case for a cohort model of supervision and support. *South Afric. J. Educ.* **31**(1), 15–30 (2017)
12. Maor, D., Currie, J.K.: The use of technology in postgraduate supervision pedagogy in two Australian universities. *Int. J. Educ. Tech. High. Educ.* **14**(1), 1–15 (2017)
13. McFarlane, J.: Group supervision: an appropriate way to guide postgraduate students? *Acta Acad.* **42**(4), 148–170 (2010)
14. Manyike, T.V.: Postgraduate supervision at an open distance e-Learning institution in South Africa. *South Afric. J. Educ.* **37**(2), 1–11 (2017)

15. Marnewick, A., Nel, H.: A model for postgraduate supervision of large student numbers in engineering management at the University of Johannesburg. In: Proceedings of IEEE Technology & Engineering Management Conference, TEMSCON, pp. 394–399 (2017)
16. Meng, Y., Tan, J., Li, J.: Abusive supervision by academic supervisors and postgraduate research students' creativity: the mediating role of leader-member exchange and intrinsic motivation. *Int. J. Leadersh. Educ.* **20**(5), 605–617 (2017)
17. Njie, B., Asimiran, S., Basri, R.: An exploratory study of the free-riding debacle in a Malaysian university: students' perspectives. *Asia Pac. Educ. Res.* **22**(3), 257–262 (2013)
18. Olmos-López, P., Sunderland, J.: Doctoral supervisors' and supervisees' responses to co-supervision. *J. Furth. High. Educ.* **41**(6), 727–740 (2017)
19. Olmos-Vega, F., Dolmans, D., Donkers, J., Stalmeijer, R.E.: Understanding how residents' preferences for supervisory methods change throughout residency training: a mixed-methods study. *BMC Med. Educ.* **15**(177), 1–8 (2015)
20. Patel, P.: An evaluation of the current patterns and practices of educational supervision in postgraduate medical education in the UK. *Perspect. Med. Educ.* **5**(4), 205–214 (2016)
21. Pringle Barnes, G., Cheng, M.: Working independently on the dissertation proposal: experiences of international master's students. *J. Furth. High. Educ.* **43**(8), 1120–1132 (2019). <https://doi.org/10.1080/0309877X.2018.1450965>
22. Roets, L., Botha, D., van Vuuren, L.: The research supervisor's expertise or postgraduate student preparedness: which is the real concern? *Afric. J. Nurs. Midwifery* **19**(2), 1–10 (2017)
23. Sidhu, G.K., Kaur, S., Chan, Y.F., Lee, L.F.: Establishing a holistic approach for postgraduate supervision. In: Tang, S.F., Logonnathan, L. (eds.) *Taylor's 7th Teaching and Learning Conference 2014 Proceedings*, pp. 529–545. Springer, Singapore (2015). https://doi.org/10.1007/978-981-287-399-6_48
24. Sidhu, G.K., Kaur, S., Choo, L.P., Fook, C.Y.: Developing a framework for postgraduate supervision. In: Teh, G.B., Choy, S.C. (eds.) *Empowering 21st Century Learners Through Holistic and Enterprising Learning*, pp. 255–267. Springer, Singapore (2017). https://doi.org/10.1007/978-981-10-4241-6_26
25. Spiller, D., Byrnes, G., Ferguson, P.B.: Enhancing postgraduate supervision through a process of conversational inquiry. *High. Educ. Res. Dev.* **32**(5), 833–845 (2013)
26. Stephens, S.: The supervised as the supervisor. *Educ. + Train.* **56**(6), 537–550 (2014)
27. University of South Africa: *Open Distance eLearning Policy*. Pretoria (2018)
28. Wichmann-Hansen, G., Thomsen, R., Nordentoft, H.M.: Challenges in collective academic supervision: supervisors' experiences from a master programme in guidance and counselling. *High. Educ.* **70**(1), 19–33 (2015)
29. van Wyk, N.C., Coetzee, I.M., Havenga, Y., Heyns, T.: Appreciation of the research supervisory relationship by postgraduate nursing students. *Int. Nurs. Rev.* **63**(1), 26–32 (2016)
30. Yin, R.K.: *Case Study Research: Design and Methods*. SAGE, Thousand Oaks (2014)



Guidelines for Conducting Design Science Research in Information Systems

Alta van der Merwe¹ , Aurona Gerber^{1,2} , and Hanlie Smuts¹ 

¹ Department of Informatics, University of Pretoria, Pretoria, South Africa
{alta,aurona.gerber,hanlie.smuts}@up.ac.za

² Centre for AI Research (CAIR), Pretoria, South Africa

Abstract. Information Systems (IS) as a discipline is still young and is continuously involved in building its own research knowledge base. Design Science Research (DSR) in IS is a research strategy for design that has emerged in the last 16 years. Junior IS researchers are often lost when they start with a project in DSR. We identified a need for a set of guidelines with supporting reference literature that can assist such novice adopters of DSR. We identified major themes relevant to DSR and proposed a set of six guidelines for the novice researcher supported with references summaries of seminal works from the IS DSR literature. We believe that someone new to the field can use these guidelines to prepare him/herself to embark on a DSR study.

Keywords: Information Systems · Design Science Research · Postgraduate students · Guidelines

1 Introduction

Design Science Research (DSR) in Information Systems (IS) has received significant attention in the last 16 years and is now accepted as an approach in top IS publication outlets such as MISQ [14]. In DSR we differentiate between design and a design theory, where design focuses on the “*use of scientific principles, technical information and imagination in the definition of a structure, machine or system to perform pre-specified functions with the maximum economy and efficiency*” and design theory is “*a prescriptive theory based on theoretical underpinnings which says how a design process can be carried out in a way which is both effective and feasible*” [52] (pp. 36–37). One of the first references in IS to the concept of ‘design science’ (DS) was in 1993 when Cross referred to DS as “*an explicitly organised, rational and wholly systematic approach to design*” [12] (p. 66). Bayazit focused on the concept of man-made things when he defined design research as a “*systematic inquiry whose goal is knowledge of, or in, the embodiment of configuration, composition, structure, purpose, value, and meaning in man-made things and systems*” [7] (p. 16). In contrast, Hevner (et al.) focus more on the practical nature of DSR when referring to design science as “*fundamentally a problem solving paradigm*” in that “*DS seeks to create innovations that define the ideas, practices, technical capabilities, and products through*

which the analysis, design, implementation, management, and use of IS can be effectively and efficiently accomplished” [22] (p. 76).

Because of the many DS and DSR discourses, novice researchers in postgraduate studies introduced to the world of research in IS have problems in making sense of the concepts. Adopting DSR as the appropriate approach to use in research requires from a researcher in-depth understanding of the literature and the progression of the field. It is however imperative to understand that there have been different viewpoints in the field, for example, on what could be considered as a research contribution, how DSR should be executed and what the underpinning philosophy of DSR is. It is important for the novice DSR researcher to take cognisance of these viewpoints, but it should also be understood that guidance is needed to assist the researcher in embarking on DSR. The purpose of this paper is therefore to contribute to the understanding of the novice researcher in DSR of the concepts on which to focus and to give an overview of the leading works that should be considered in preparing to embark on a DSR research project.

In this paper we will next discuss how we conducted our research in Sect. 2, followed by the suggested guidelines in Sect. 3, before proceeding to an overview of the different concepts to be consulted by the novice researcher or postgraduate student. We conclude the paper in Sect. 4 with some suggestions for future work.

2 Method

The focus of this paper is on giving guidelines and discussing some of the concepts that we believe are of importance to the novice researcher or postgraduate student. We followed a two-phase approach to answer the *research questions* listed in Table 1.

Table 1. Research questions

	RESEARCH QUESTIONS	DATA COLLECTION
RQ1	What are the guidelines supervisors give to novice DS researchers embarking on a new DSR project?	Focus group
RQ2	Who are the key DSR research leaders to consult for the different concepts identified in RQ1?	Literature review
RQ3	What are the seminal works that should be considered by a novice DS researcher?	Literature review

Our two-phase approach (Subsects. 3.1 and 3.2) consists of involving a focus group to answer the first research question and a systematic literature review in the second phase to answer the second and third research questions.

For the first phase, the focus group, we used the guidelines provided by Barber and Rossi [3] with three experienced DSR supervisors selected by convenience

Table 2. Themes for conducting DSR

FOCUS AREA	THEME	DESCRIPTION
Positioning of DSR	Artefact	<i>“Design science products are of four types: constructs, models, methods, and implementations”</i> [29]
	Relevance, Rigour, Practice	This theme focuses on the discussion of DSR as a practice (relevance), but also contributes to existing theory (rigour) [22]
	Design Theory	Design theories are also seen as a product of DSR by several authors [4], and emerged as theme
Research Design	Philosophy	In conducting research, the ontological stance of a researcher is discussed during research design
	Method	The method followed during DSR was one of the first focus areas in the development of DSR as a field [45]
Communication	Argument	This theme relates to how a researcher communicates the research to the research community
	Thesis (Research Report)	This theme relates to practices for sharing the processes of the DSR and the new knowledge related to the creation of the artefact or the nature of the artefact

sampling from our university. The supervisors have been collectively involved in supervision of 26 PhD and Master students who used the DSR approach in their projects. The focus group was conducted as a group interview with the goal to capture the way in which the supervisors guide the researcher new to DSR in finding his/her way in order to do a DSR study. The summarised notes were analysed with two goals: firstly to identify the themes (Table 2) and secondly to identify the guidelines (Table 3) linked to the themes on conducting DSR research. After the themes were identified by the focus group, a short survey was sent out to 22 experienced supervisors at other universities to confirm the themes. There was a response from 13 supervisors from 9 universities where the themes were confirmed with all of them—indicating that the DSR process is the most important theme.

The second phase of our project was to identify the research leaders in DSR, linked to the themes identified in the first phase, and to ensure that we were able to give guidance in this paper on the seminal works linked to the themes. We followed the steps of a systematic literature review with the goal to describe available knowledge. This is in line with Okoli who states that *“one of the reasons for conducting a systematic literature review is to describe available knowledge for professional practice”* [34] (p. 82). An eight-step process was followed in

Table 3. DSR guidelines for novice researchers or postgraduate students

	GUIDELINE
1.	Contextualise DSR in the field of Information Systems and be able to distinguish between concepts such as design, design science and DSR
2.	Understand the philosophical underpinning of research and the discourse on the nature of DSR
3.	Obtain a historical perspective of DSR and consult the work of the pioneers in the field
4.	Consider the role of the artefact in DSR and the different views on design theory
5.	Select an appropriate DSR method for execution of the research study
6.	Strategise on how research done in DSR should be communicated in a report such as a thesis

the review according to [34], including: (1) identifying the purpose, (2) drafting protocol, (3) applying practical screen, (4) searching for literature, (5) extracting data, (6) appraising quality, (7) synthesising studies and (8) writing the review.

For the first step, identifying the purpose, the research questions were used as guideline. The draft protocol was compiled together with the application of the practical screen, where the procedure was discussed that would be used during the systematic literature review. The search terms identified included the following terms (and combinations of the terms), ‘design science’, ‘design science research’, ‘design research’, and ‘information systems’. During the fourth step, searching for literature, we started with the ‘basket of eight’ in IS [1], followed by searches for publications in DESRIST which hosted a conference every year since 2006 that focus on DSR in IS.¹ We followed an iterative process during the search process: if a publication in later years referenced earlier works that were not in the initial set, these were also included. This extended the documents to include material from other sources not listed above. We excluded works from other fields, such as Education, Engineering and Economic and Management Sciences, since our focus was specifically only on IS. We acknowledge that there might be valuable resources available in these fields, but we believe that this opens up a new research topic where future research is possible to see how the different fields align, especially from a practice point of view. We did not include papers focusing only on DSR examples or case studies—all papers contributed to the themes identified in the first phase (focus group) of the data collection. In total 124 papers were identified, which were captured in an Excel spreadsheet and included in the remaining analysis. Our next step was to extract the data, where the extraction consisted of doing a Google scholar classification for each paper to indicate the citation as in February 2019 (this information was used to identify the most referenced papers) and then the papers were sorted according

¹ <http://desrist.org/about/>.

to citation value. The next step was to appraise the quality, where each paper was classified according to the themes identified in the focus group sessions and papers that did not align to one of the themes were excluded. We synthesised the studies by firstly grouping together studies that focused on specific themes with high citations and then as a second step considering papers with lower citations that focused on topics relevant to the themes identified for DSR. The last step was to communicate the results of the research, as done in this paper.

3 An Information Systems Design Science Research Roadmap

3.1 Phase 1: DSR Guidelines and Themes

The first phase of the data analysis was based on the data collected during the focus group session. Firstly, seven themes were identified as pertinent in DSR for a novice researcher. These themes were categorised into three broader focus areas, including the positioning of DSR, the research design and communication (Table 2).

For the focus area, positioning of DSR, the focus is on the artefact, the relevance and rigour of creating the artefact, and the design theory. A second focus area relates to the design of the research and focuses on the philosophy and the method (or process) followed. The last focus area relates to the communication of the design process followed, where a researcher should focus on the argument and guidelines relating to structuring a thesis or publication. After identification of the focus areas and themes, guidelines were identified that would help a novice researcher or postgraduate student to conduct DSR (Table 3).

3.2 Phase 2: Relevant DSR Content According to Guidelines and Themes

Here we discuss the literature that was identified during the systematic literature review according to the themes and the guidelines identified.

Guideline 1: *Contextualise DSR in the field of Information Systems and be able to distinguish between concepts such as design, design science and DSR.*

In IS, novice researchers are exposed to different research directions either by supervisors or more formally in courses taken by students as part of their preparation for a research project. In a research project a researcher will typically start exploring the problem, read the literature and explore different directions to conduct the research, depending on what the researcher has been exposed to or guidance given by a mentor. During this phase the researcher might consider DSR if (s)he is involved in the process of design.

DSR is often discussed from the perspective of the science of the artificial, as done by Simon [42], who introduced the notion that one can study the artefact as part of science in 1969. We acknowledge that the concept of design was used

in other fields, such as engineering [51], but in IS the work of Simon as originally written in 1969 and revised later editions [42] is cited by many authors as a seminal work. Gregory argues that in doing design one is creating something that does not yet exist [19]. There are two concepts of importance in this argument—there is creation of something (the artefact), and there is the process of creation. Design is therefore “*both a noun and a verb*” [19] (p. 3), or a process and a product. In 1992 Walls emphasised that we as IS practitioners and IS users have been involved in the process of design for several years through systems development [52]. As mentioned in the introduction, Cross in 1993 described DS as “*a systematic approach to design*” [12]. In the same year Smith and Browne also focused on the topic of DS and emphasised the difficulties in design due to human involvement [43]. They argue that Simon’s view in [42] was a DS view, although Simon never used the term DS. Simon referred to the “*science of design*”. According to [43], DS should focus on understanding the designer as well as on the processes to be used for design. Another view is that of March and Smith, who contrast natural science and DS and argue that DS is “*concerned with the creation of artefacts to attain goals that serve human purposes*” [29] (p. 253).

DSR in IS reached a milestone in 2004 when Hevner (et al.) presented their framework for IS research and guidelines for DSR [22]. In that work they referred to DSR as a paradigm where the “*knowledge and understanding of a problem domain and its solution are achieved in the building and application of the designed artefact*” [22] (p. 75). More or less in the same timeframe Vaishnavi (et al.) started a web site focusing on DSR in IS [48]. According to them, “*DSR uses a set of synthetic and analytical techniques and perspectives for performing research in IS*”. Furthermore, they define “*DSR as being involved in the creation of new knowledge, firstly through the development of artefacts and secondly through the study of the use of the artefact afterwards*”.

Guideline 2: *Understand the philosophical underpinning of research and the discourse on the nature of DSR.*

In conducting the data collection on the ‘philosophy’ theme, only works were included that explicitly discuss the philosophical stand of DSR. Research conducted in IS is mostly multi-disciplinary and the philosophy mostly found is either positivist, interpretivist, or critical research. In the papers reviewed, three discourses emerged, including (1) DSR as paradigm, (2) Traditional paradigms, and (3) Pragmatism: see below.

DSR as Paradigm. Originally Vaishnavi (et al.) discussed DSR as a paradigm on its own [48]. They argued that design can be research and that it changes the world through the development of new artefacts. Their initial ideas were shared on a website hosted by DESRIST and later replicated in their book [47] in which they contrast interpretivism, positivism, and DSR in tabular form. We summarise their table in Table 4, as a partial view of their comparison, to show how DSR is described.

Cross also argues for the recognition of DSR as discipline [11]: he states that we can have discussions on design and the value of the creative activity and

Table 4. Philosophical assumptions of DSR, taken from [48]

ONTOLOGY	EPISTEMOLOGY	METHODOLOGY	AXIOLOGY
Multiple, contextually situated alternative world-states. Socio-technologically enabled	Knowing through making; objectively constrained construction within a context. Iterative circumscription reveals meaning	Developmental Measure artefactual impacts on the composite systems	Control; creation; problem-solving; progress (i.e. improvement); understanding

share experiences of the process. He further argues that designers understand and know the artificial world and know how to change and add to this world.

Traditional Paradigms. In the second discourse on philosophical grounding of DSR, arguments are provided for the use of philosophies traditionally used in IS, such as interpretivism or positivism. Gregory claims that “*DSR is conducted most frequently within a positivistic epistemological perspective*” [19]. Venable (et al.) propose a framework for understanding design research where the framework focuses on theory building as well as evaluation of the solutions from a positivist or interpretivist angle [49]. Carlsson proposes a framework of IS DSR with the aim to develop practical knowledge for the design and realisation of IS initiatives including socio-technical systems [9]. His underpinning philosophy of the framework is critical realism. Critical realism’s aim is to “*recognize the reality of the natural order and the events and discourses of the social world*”. It holds that we will only be able to understand—and so change—the social world if we identify the structures at work that generate those events or discourses [9] (p. 200).

Pragmatism as Paradigm. March and Smith were some of the first authors to emphasise pragmatism when they argued that truth is what works in practice [29]. In 2007, Hevner devoted the closure of his article to claiming pragmatism as the nature of DSR [20]. His view of pragmatism is that it is a “*school of thought that considers practical consequences or real effects to be vital components of both meaning and truth*” [20] (p. 93). He argues that the synergy between practical and theoretical contributions is what defines good DSR. His view is confirmed in later papers [15, 21, 28]. A useful source on the nature of DSR is the paper by Goldkuhl [15]. In this seminal work he investigates the epistemological foundation for design research and argues that the pragmatist perspective is fit for DSR based on its focus on utility and knowledge growth through development, starting with a problematic situation and aiming for knowledge (by) building. More recently Deng and Ji argued that pragmatism is the underpinning philosophy for DSR [13] but does not exclude different phases wherein a researcher is involved as interpretivist, positivist and constructive observer or intervener (Fig. 1).

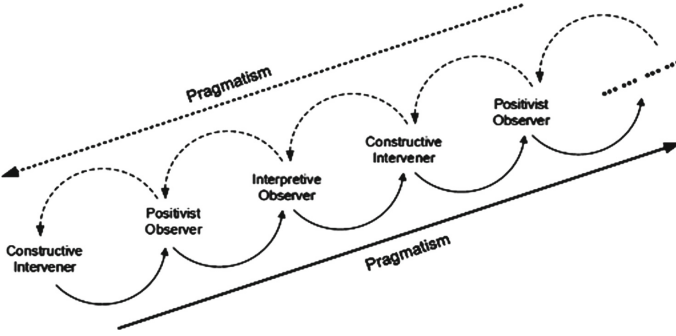


Fig. 1. Iterative design science process according to [13]

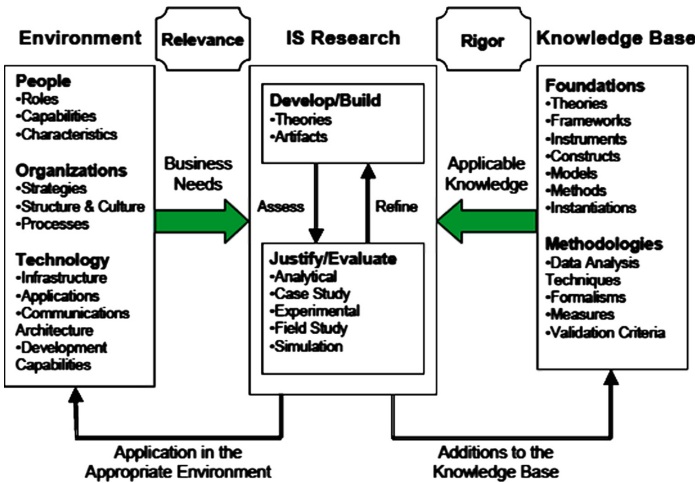


Fig. 2. IS research framework according to [22]

Guideline 3: Obtain a historical perspective of DSR and consult the work of the pioneers in the field.

As mentioned previously, the field of DSR evolved much earlier in other fields such as engineering and architecture. The most frequently cited work in IS is the work of Simon [42] wherein he argues for the acceptance of the study and development of artificial or man-made objects. He also refers to problems experienced in management in the field of IS.

Following the work of Simon [42], the seminal work of Hevner (et al.) [22] from the year 2004 was most highly cited; in it they contrasted behavioural science and DS, and presented a framework (Fig. 2) for IS research together with a set of guidelines for DSR.

Hevner (et al.) argue that IS research has the dual value of rigour and relevance [22]. On the rigour side (Fig. 2), the researcher gets applicable knowledge

Table 5. DSR guidelines according to [22]

	GUIDELINE	ADVICE
1.	Design as Artefact	Design science research must produce a viable artefact in the form of a construct, a model, a method, or an instantiation
2.	Problem Relevance	The objective of design science research is to develop technology-based solutions to important and relevant business problems
3.	Design Evaluation	The utility, quality, and efficacy of a design artefact must be rigorously demonstrated via well-executed evaluation methods
4.	Research Contributions	Effective design science research must provide clear and verifiable contributions in the areas of the design artefact, design foundations, and/or design methodologies
5.	Research Rigour	Design science research relies upon the application of rigorous methods in both the construction and evaluation of the design artefact
6.	Design as Search	The search for an effective artefact requires utilising available means to reach desired ends while satisfying laws in the problem environment
7.	Communication of Results	Design science research must be presented effectively to both technology-oriented and management-oriented audiences

from the knowledge base, including existing theories, frameworks etc. On the relevance side the need for a new artefact arises, articulated as *business needs* (Fig. 2). Business needs from the environment can stem from people, technology, or organisations. In the centre are the activities related to development, building and evaluation of the new artefact. At the bottom of Fig. 2 the contribution is both back to the environment in the form of an artefact with practical value and to rigour in the form of new knowledge. A further contribution of [22] are the guidelines summarised in Table 5.

Prior to [22], three papers were published in the 1990s that led to significant citations. These included that by March and Smith [29], which proposed the four types of artefacts referenced in later years by several authors, that of Walls (et al.) [52], which focused on the creation of a design theory (Guideline 4), and that of Nunamaker (et al.) [32], which proposed to conduct design research based on the system analysis and design method (Guideline 5). In 2007 Gregor and Jones built on [52] in design theory (Guideline 4) and distinguished between a product and a process artefact [18]. Gregor and Hevner elaborated on the nature of design research [17], and provided a guide for reporting on and communicating DSR (Guideline 6). These papers, which are regarded as seminal works, are summarised in Table 6.

Table 6. Seminal publications in DSR

REF.	#CIT.	YEAR	SIGNIFICANCE
[29]	3979	1995	Initially proposed types of artefacts
[52]	1530	1992	Focus on design theory; method for theory building
[32]	1508	1991	Proposes method; argues from system development background for design
[18]	1428	2007	Focus on design theory; distinguishes between two different kinds of purposeful artefacts that can be designed: product artefacts and process artefacts
[17]	1402	2013	DSR overview; positions DSR; gives guidance on publishing
[11]	1306	2001	Nature of DSR; distinguishes between scientific design, design science, a science of design
[30]	1282	2002	Example of a design theory for knowledge management processes

More recent work with fewer citations that serves as a good starting point in understanding the concepts in DSR has been published by Baskerville (et al.) [4], and Deng and Ji [13].

In the early days of DSR many authors argued that DSR and action research (AR) would be the same. A novice researcher needs to take cognisance of these discussions to be able to understand that, though there are similarities, they are not the same. Here we recommend Iivari [23, 24] and Sein (et al.) [41]. Another contribution on the topic of AR and DSR is the work by Lee that combines action and design research methods into a single framework for design [27].

Guideline 4: *Consider the role of the artefact in DSR and the different views on design theory.*

Central to DSR is the artefact or an artificial and man-made object. The first mention of different types of artefacts is by March and Smith as constructs, models, methods and implementations [29]. Winter gives examples of constructs that include modelling primitives implemented by meta-models of modelling tools, process models implemented as workflows, models and project methods used during software package introduction as a method [53]. Puroo claimed in 2002 that the artefact created in DSR is software or a system [37]. Hevner and Chatterjee as well as Vaishnavi (et al.) also give as examples of the artefact algorithms, human/computer interfaces, languages, and system design methodologies [21, 48]. In 2010 Offerman (et al.) wrote a literature review on the types of artefacts in IS design science and suggested a topology with eight types of artefacts [33]: these included a system design, method, language (notation), algorithm, guideline, requirements, pattern and metric.

In 2003, Rossi and Sein (in acknowledged collaboration with Puroo) added ‘better theories’ as artefacts [40], however, not all experts agreed. Winter argued

that, although theory building is not design science research, theories as *intermediate* artefacts need to be included in the system of relevant artefacts for IS design science research [53] (p. 472). Baskerville (et al.) emphasised that DSR brings about both practical relevance by developing useful artefacts and scientific rigour by the formulation of design theories [4].

The topic of design theories was discussed in the early introduction of DSR into IS. Many of the later publications build on the work of Walls (et al.) who distinguished between a design product and a design process in their classification of the components of an information systems design theory (ISDT) [52]. They characterise design theories as (1) dealing with goals as contingencies, (2) never involving pure explanation or prediction, (3) being prescriptive, (4) being composite theories that encompass kernel theories from natural science, social science and mathematics. They claim that whereas “*explanatory theories tell what is, predictive theories tell what will be, and normative theories tell what shall be, design theories tell how to/because*” [52] (p. 40). It should be noted that Walls (et al.) regard ‘theory’ as the design of an artefact and the method followed. This is evident when they propose ISDT as an output of design science.

Gregor contributed to the discussion on theory by defining five classes of theory [16]. Design theory is the last of this set of classes, which includes “(1) *theory for analysing, (2) theory for explaining, (3) theory for predicting, (4) theory for explaining and predicting, and (5) theory for design and action*”. In their seminal work on design theory published in 2007, Gregor and Jones emphasised that “*we need to pay attention to how design knowledge is expressed as theory*” [18]. They extended the work of [52] and identified eight separate components of design theories.

Theory development will remain topical in DSR and several publications are recommended, such as the work by Kuechler and Vaishnavi [26], and Baskerville and Pries-Heje [5]. Also [4] should be considered as it reflects on the balance between contributions in science (theory) and technology (artefacts). Accordingly, in DSR some degree of design theorising should be expected, where the initial conceptualisation of the artefact is the first step in theorising; however, design theory (prescriptive, scientific knowledge) is a desirable goal as theorising around a class of artefacts progresses [4] (p. 369).

Guideline 5: *Select an appropriate DSR method the research project.*

Originally March and Smith argued that design science consists of two basic activities, namely building and evaluating [29]. Here we therefore give an overview of the subsequent pertinent works with regard to the methodology for DSR construction, and then discuss the evaluation of DSR.

All methods described in the literature on conducting DSR consist of a combination of the general design and development phases, namely identification, design, development and testing. Vaishnavi (et al.) published one of the often used and referenced methods that they call a DSR *process model* [48], which was based on [45] and is illustrated in Fig. 3. In this model they illustrate that a DSR project goes through cycles of awareness, suggestion, development, evaluation,

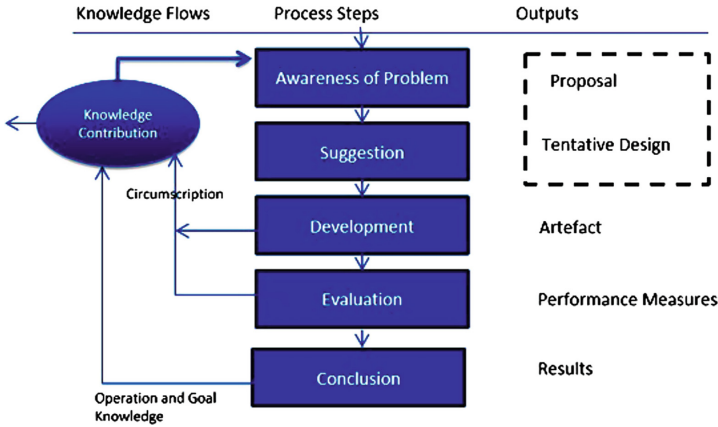


Fig. 3. DSR process model according to [22]

and conclusion. The knowledge or theory contribution is through circumscription illustrated on the left-hand side as an exit point to development, evaluation or conclusion. They also argue that the outputs for each phase range from the proposal during awareness, the tentative design during the suggestion phase, the artefact during awareness, performance measures for the evaluation and then lastly the results in the conclusion.

Another popular DSR process model often used is the work by Peffers (et al.) [36]. In their process model (Fig. 4) the DSR cycles through “*problem identification and motivation, objectives of solution, design and development, demonstration, evaluation and communication*”. They provide for different entry points into the process model, depending on the type of development to be conducted. It might be that one has an existing artefact that needs refinement, which will not necessarily need to go through all the phases, but might for example enter only at the design and development phase.

Other significant publications on methods for DSR include Baskerville (et al.) wherein they propose a seven-phase ‘soft’ DS methodology [6], vom Brocke and Buddendic who that suggest that the DSR cycle consists of six phases [8], as well as Alturki (et al.) [2]. Vahidov presented an innovative way of developing the artefact [46] based on Zachman’s Framework [54].

For the evaluation of the artefact, the pioneers working in this field were Pries-Heje, Baskerville and Venable, who published several papers [38,39,49] building up towards their framework for evaluation in design science, FEDS [50]. The FEDS was designed to assist DSR researchers in deciding on a way to evaluate the outcomes during development. They highlight two dimensions in their framework, namely the “*functional purpose of the evaluation (formative or summative) and the paradigm of the evaluation (artificial or naturalistic)*”. In their framework they identified four different possible strategies, namely the “*quick and simple strategy, the human risk and effectiveness evaluation strat-*

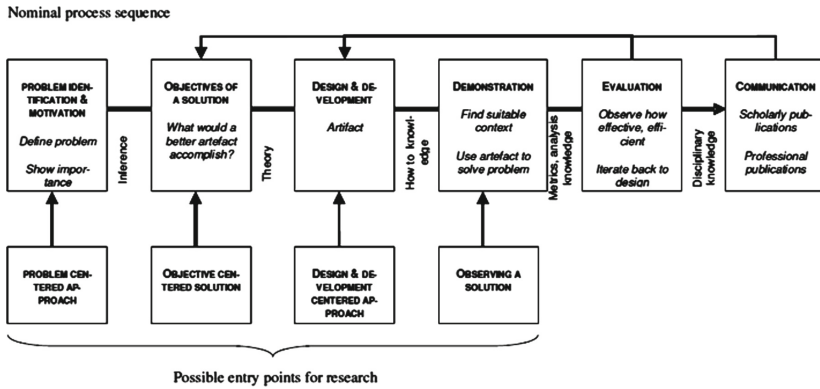


Fig. 4. DSR process model according to [36]

egy, the technical risk and efficacy evaluation strategy, and the purely technical artefact strategy”. Accordingly they provided a four-step process for choosing an approach for a particular DSR, namely: (1) explicate the goals of the evaluation, (2) choose the evaluation strategy or strategies, (3) determine the properties to be evaluated, and (4) design the individual evaluation episode(s).

Other significant work on evaluation includes Cleven (et al.), Peffers (et al.), as well as Sonnenberg and vom Brocke [10, 35, 44]. A ‘roadmap’ to conduct DS research was published by Alturki (et al.) [2] which adopts the three DS research cycles of [20], namely: rigour, relevance, and design. Their contribution is a 14-step procedure that novices can follow to do DSR.

Guideline 6: *Strategise on how DSR research (results) should be communicated in a report (paper or thesis).*

The last guideline for DSR is applicable when one needs to strategise on how to communicate research results. Gregor and Hevner give advice on publishing papers in DSR [17], whereby they propose a publication schema for recording results. They argue that the four questions that reviewers will ask are whether the problems discussed in a paper are of substantial interest, whether the problems are solved or a contribution is made to a solution, whether the methods are new, and whether a paper increases understanding of the area of research.

Kotze (et al.) used the guidelines of [22] for DSR and commented on questions to be asked for each of the guidelines [25]. Some of the considerations are: to be clear from the start about the type of artefact that will be designed, to reconsider the uniqueness of the artefact, to think about how one will do the evaluation, what the contribution will be, how one will collect information needed to ‘build’ the artefact or evaluate the artefact, and what the value of the artefact is.

In [31] we described a method that a student can use to write a thesis in DSR according to the steps of [48]. We argue that the introduction and literature review of a thesis map to the ‘awareness phase’, the literature review and

research design map to the ‘suggestion phase’, the research design and body of the thesis give an overview of ‘development’, while the body of the thesis should also describe the ‘evaluation phase’. The last phase, ‘conclusion’, will then be presented in the conclusion of a thesis [31].

4 Conclusion

In this paper we provide an overview of DSR as a guide for a novice IS researcher embarking on a DSR project. After having identified the major themes relevant to a DSR project, and after having proposed a set of six guidelines for the novice researcher, we corroborated our guidelines by referring to the seminal works of the DSR field.

We believe that the value of this paper is two-fold. Firstly, a researcher unfamiliar with the field can follow our guidelines to prepare him/herself for a DSR project. Secondly, the seminal DSR works to date (within IS) are listed and summarised such as to serve as a *reference guide* for postgraduate students.

References

1. AIS: Information Systems Basket of Eight (2019). <https://aisnet.org/general/custom.asp?page=SeniorScholarBasket>
2. Alturki, A., Gable, G.G., Bandara, W.: A design science research roadmap. In: Jain, H., Sinha, A.P., Vitharana, P. (eds.) DESRIST 2011. LNCS, vol. 6629, pp. 107–123. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-20633-7_8
3. Bader, G.E., Rossi, C.A.: Focus Groups: a Step-by-Step Guide. Bader Gr. (1999)
4. Baskerville, R., Baiyere, A., Gergor, S., Hevner, A., Rossi, M.: Design science research contributions: finding a balance between artifact and theory. *J. Assoc. Inf. Syst.* **19**(5), 358–376 (2018)
5. Baskerville, R., Pries-Heje, J.: Explanatory design theory. *Bus. Inf. Syst. Eng.* **2**(5), 271–282 (2010)
6. Baskerville, R., Pries-Heje, J., Venable, J.: Soft design science methodology. In: Proceedings of DESRIST 4th International Conference on Design Science Research in Information Systems and Technology. ACM (2009)
7. Bayazit, N.: Investigating design: a review of forty years of design research. *Des. Issues* **20**(1), 16–29 (2004)
8. vom Brocke, J., Buddndick, C.: Reusable conceptual models – requirements based on the design science research paradigm. In: Proceedings of DESRIST. Springer (2006)
9. Carlsson, S.A.: Towards an information systems design research framework: a critical realist perspective. In: Proceedings of DESRIST, p. 21 (2006)
10. Clevén, A., Gubler, P., Höner, K.M.: Design alternatives for the evaluation of design science research artifacts. In: Proceedings of DESRIST. ACM (2009)
11. Cross, N.: Designerly ways of knowing: design discipline versus design science. *Des. Issues* **17**(3), 49–55 (2001)
12. Cross, N.: Science and design methodology: a review. *Res. Eng. Des.* **5**(2), 63–69 (1993)



13. Deng, Q., Ji, S.: A review of design science research in information systems: concept, process, outcome, and evaluation. *Pac. Asia J. Assoc. Inf. Syst.* **10**(1), 36 (2018)
14. Goes, P.B.: Design science research in top information systems journals. *MIS Q.* **38**(1), iii–viii (2014)
15. Goldkuhl, G.: Design research in search for a paradigm: pragmatism is the answer. In: Helfert, M., Donnellan, B. (eds.) *EDSS 2011. CCIS*, vol. 286, pp. 84–95. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33681-2_8
16. Gregor, S.: The nature of theory in information systems. *MIS Q.* **30**(3), 611 (2006)
17. Gregor, S., Hevner, A.R.: Positioning and presenting design science research for maximum impact. *MIS Q.* **37**(2), 337–355 (2013)
18. Gregor, S., Jones, D.: The anatomy of a design theory. *J. Assoc. Inf. Syst.* **8**(5), 312–335 (2007)
19. Gregory, R.W.: Design science research and the grounded theory method: characteristics, differences, and complementary uses. In: Heinzl, A., Buxmann, P., Wendt, O., Weitzel, T. (eds.) *Theory-Guided Modeling and Empiricism in Information Systems Research*, pp. 111–127. Physica-Verlag, Heidelberg (2011)
20. Hevner, A.R.: A three cycle view of design science research. *Scand. J. Inf. Syst.* **19**(2), 87–92 (2007)
21. Hevner, A.R., Chatterjee, S.: Design science research in information systems. In: Hevner, A.R., Chatterjee, S. (eds.) *Design Research in Information Systems*, pp. 9–22. Springer, Boston (2015). https://doi.org/10.1007/978-1-4419-5653-8_2
22. Hevner, A.R., Ram, S.M., Park, J.: Design science in information systems research. *MIS Q.* **28**(1), 75–105 (2004)
23. Iivari, J.: Distinguishing and contrasting two strategies for design science research. *Eur. J. Inf. Syst.* **24**(1), 107–115 (2015)
24. Iivari, J.: Information systems as a design science. In: Vasilecas, O., Wojtkowski, W., Zupančič, J., Caplinskas, A., Wojtkowski, W.G., Wrycza, S. (eds.) *Information Systems Development*, pp. 15–27. Springer, Boston (2005). https://doi.org/10.1007/0-387-28809-0_2
25. Kotze, P., van der Merwe, A., Gerber, A.: Design science research as research approach in doctoral studies. In: *Proceedings of AMCIS (2015)*
26. Kuechler, W., Vaishnavi, V.: A framework for theory development in design science research: multiple perspectives. *J. Assoc. Inf. Syst.* **13**(6), 29 (2012)
27. Lee, A.S.: Action is an artifact. In: Kock, N. (ed.) *Information Systems Action Research*, pp. 43–60. Springer, Boston (2007). https://doi.org/10.1007/978-0-387-36060-7_3
28. Levy, M., Hirschheim, R.: Removing the positivist straight jacket from information systems design science research. In: *Proceedings of ECIS*, p. 13 (2012)
29. March, S.T., Smith, G.F.: Design and Natural Science Research on Information Technology. *Decis. Support Syst.* **15**(4), 251–266 (1995)
30. Markus, L., Majchrzak, A., Gasser, L.: A design theory for systems that support emergent knowledge processes. *MIS Q.* **26**(3), 179–212 (2002)
31. van der Merwe, A., Gerber, A., Smuts, H.: Mapping a design science research cycle to the postgraduate research report. In: Liebenberg, J., Gruner, S. (eds.) *SACLA 2017. CCIS*, vol. 730, pp. 293–308. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-69670-6_21
32. Nunamaker, J.F., Chen, M., Purdin, T.D.M.: Systems development in information systems research. *J. Manag. Inf. Syst.* **7**, 89–106 (1991)

33. Offermann, P., Blom, S., Schönherr, M., Bub, U.: Artifact types in information systems design science – a literature review. In: Winter, R., Zhao, J.L., Aier, S. (eds.) DESRIST 2010. LNCS, vol. 6105, pp. 77–92. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13335-0_6
34. Okoli, C.: A guide to conducting a standalone systematic literature review. *Commun. Assoc. Inf. Syst.* **37**, 879–910 (2015)
35. Peffers, K., Rothenberger, M., Tuunanen, T., Vaezi, R.: Design science research evaluation. In: Peffers, K., Rothenberger, M., Kuechler, B. (eds.) DESRIST 2012. LNCS, vol. 7286, pp. 398–410. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29863-9_29
36. Peffers, K., Tuunanen, T., Rothenberger, M.A., Chatterjee, S.: A design science research methodology for information systems research. *J. Manag. Inf. Syst.* **24**(3), 45–77 (2007)
37. Purao, S.: Design research in the technology of information systems: truth or dare. Technical report, GSU Department of CIS (2002)
38. Pries-Heje, J., Baskerville, R.: The design theory nexus. *MIS Q.* **32**(4), 731 (2008)
39. Pries-Heje, J., Baskerville, R., Venable, J.R.: Strategies for design science research evaluation. In: Proceedings of ECIS (2008)
40. Rossi, M., Sein, M.K.: Design research workshop: a proactive research approach. Technical report (2003)
41. Sein, M.K., Henfridsson, O., Purao, S., Rossi, M., Lindgren, R.: Action design research. *MIS Q.* **35**(1), 37 (2011)
42. Simon, H.A.: *The Sciences of the Artificial*, 3rd edn. MIT Press, Cambridge (1996)
43. Smith, G.F., Browne, G.J.: Conceptual foundations of design problem solving. *IEEE Trans. Syst. Man Cybern.* **23**(5), 1209–1219 (1993)
44. Sonnenberg, C., vom Brocke, J.: Evaluation patterns for design science research artefacts. In: Helfert, M., Donnellan, B. (eds.) EDSS 2011. CCIS, vol. 286, pp. 71–83. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33681-2_7
45. Takeda, H., Veerkamp, P., Tomiyama, T., Yoshikawa, H.: Modeling design processes. *AI Mag.* **11**(4), 12 (1990)
46. Vahidov, R.: Design researcher’s is artifact: a representational framework. In: Proceedings of DESRIST. Springer (2006)
47. Vaishnavi, V., Kuechler, W.: *Design Science Research Methods and Patterns: Innovating Information and Communication Technology*. CRC Press, Boca Raton (2015)
48. Vaishnavi, V., Kuechler, B., Petter, S.: *Design Science Research in Information Systems* (2004). <http://desrist.org/design-research-in-information-systems/>
49. Venable, J., Pries-Heje, J., Baskerville, R.: A comprehensive framework for evaluation in design science research. In: Peffers, K., Rothenberger, M., Kuechler, B. (eds.) DESRIST 2012. LNCS, vol. 7286, pp. 423–438. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29863-9_31
50. Venable, J., Pries-Heje, J., Baskerville, R.: FEDS: a framework for evaluation in design science research. *Euro. J. Inf. Syst.* **25**(1), 77–89 (2016)
51. Vincenti, W.: *What Engineers Know and How They Know It: Analytical Studies from Aeronautical History*. Johns Hopkins University Press, Baltimore (1990)
52. Walls, J.G., Widmeyer, G.R., El Sawy, O.A.: Building an information systems design theory for vigilant EIS. *Inf. Syst. Res.* **3**(1), 36–59 (1992)
53. Winter, R.: Design science research in Europe. *Eur. J. Inf. Syst.* **17**(5), 470–475 (2008)
54. Zachman, J.: About the Zachman Framework (2008). <https://www.zachman.com/about-the-zachman-framework>

Our Students, Our Profession



Making Sense of Unstructured Data: An Experiential Learning Approach

Sunet Eybers^(✉)  and Marie J. Hattingh 

Department of Informatics, University of Pretoria, Pretoria, South Africa
{sunet.eybers,marie.hattingh}@up.ac.za

Abstract. The need for competent data scientists is recognised by industry practitioners worldwide. Currently tertiary education institutions focus on the teaching of concepts related to structured data (fixed format), for example in database management. However, the hidden value contained in unstructured data (no fixed format) motivated the need to introduce students to methods for working with these data sets. Therefore, an experiential learning approach was adopted to expose students to real-life unstructured data. Third year students were given an assignment whereby they could use any publicly available un-structured data set or an unstructured dataset supplied to them following a set methodology (CRISP-DM) to discover and describe the hidden meaning of the data. As part of the assignments students had to reflect on the process. Twenty student assignments were analysed in an attempt to identify the effectiveness of the experiential learning approach in the acquisition of skills pertaining to unstructured data. Our findings indicate that the experiential learning approach is successful in the teaching of the basic skills needed to work with unstructured data. We discuss the appropriateness of the prescribed methodology, the students' performance, and lessons learnt. On the basis of these lessons we conclude with some recommendations for educating future data scientists.

Keywords: Experiential learning · Big data · Unstructured data · CRISP-DM methodology · Data scientists

1 Introduction

Data has always been a key asset to organisations. Nowadays this asset has become even more important due to the potential value contained in big datasets [11, 33]. *Big Data* refers to data with unique characteristics such as *the three V*: volume, velocity and variety [12, 25, 33]. Some scholars even include an additional characteristic, namely value [11, 33, 36]. Volume refers to the size and subsequent quantity of data sets which are often measured in terabytes or even petabytes [16, 33]. Velocity refers to the continuous generation of data by applications such as social media whilst variety refers to different kinds of data such as operational data from various business systems, xml files and text messages [16, 33].

These different kinds of data are further classified as structured (fixed format), semi-structured (consisting of both fixed format and free text or no fixed format data), and unstructured (no fixed format) [33]. Value refers to the untapped potential worth of the meaning hidden in large data sets [12], which might be economically or financially significant [11,36]. Unfortunately, unleashing the value contained in these data sets can be challenging due to reasons pertaining to technologies, processes and human aspects [12]. For example, working with technologies such as advanced data mining tools requires specialised statistics tools; processes to combine data sources from various locations might be unclear; data scientists working with big datasets require a *combination* of business-, technical and analytical skills which is rarely found in today's human resources [34].

Despite the current scarcity of data scientists the position is said to be a "*most exciting career opportunity of the 21st century*" with above-average remuneration packages [2]. The demand for data scientists and data engineers is projected to grow with 39% [27]. The challenge from an educational perspective is to ensure that students studying in the areas of informatics, information science and computer science (ICT) are ready to meet the demands of industry practitioners upon graduation [3,32]. Although the majority of educational institutions currently focus on skills to work with data, the curriculum has a strong focus on working with structured data such as databases, data marts and data warehouses; for comparison see [20,21].

A current challenge in the curriculum, in particular the institution under study, is to introduce students to ways and methods of working with unstructured data obtained from social media platforms. Also, students often, as part of their post-graduate research projects (or fourth-year projects), are faced with challenges to work with unstructured data—a skill they have often not been exposed to during undergraduate studies. Therefore, our third-year semester module aimed at introducing students to working with unstructured data from social media platforms. A set methodology was prescribed to guide students through the assignment (namely the CRISP-DM Cross-Industry Standard Process for Data Mining explained later in this paper).

We followed an experiential learning approach where students could select their own set of unstructured data from any social media source and subsequently any tool or technique to extract meaning from those data. Our aim was to evaluate how effective the learning process was. The research question was: *how effective is an experiential learning approach in the teaching of basic skills to work with unstructured data?*

This paper continues with a brief introduction to the experiential learning approach, followed by related work focused on the topic of data science education. The CRISP-DM methodology is explained followed by a description of the case study and proposed research method. The analysis and discussion section describes the findings after evaluating the student assignments in relation to the phases of the experiential learning approach and the six steps of CRISP-DM. The discussion also includes the lecturers' reflection.

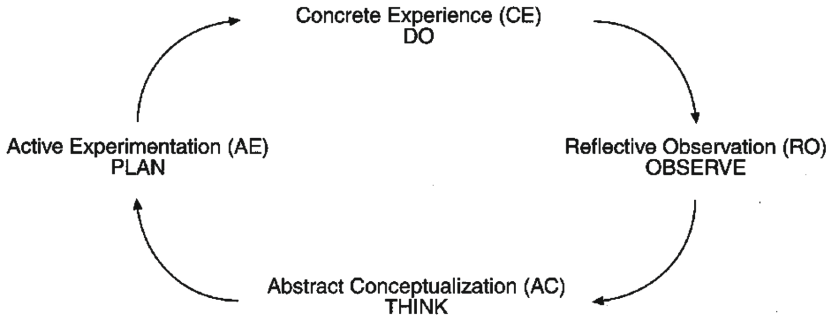


Fig. 1. Kolb's experiential learning cycle [15]

2 Experiential Learning

Experiential learning theory was introduced by Kolb in 1984 [18] and widely adopted in various educational environments [19] across fields such as medical and health [14], information systems [10], and marketing [4] (to name a few). The theory postulated that learners acquire new knowledge through practically completing tasks, i.e. their experience of interaction with the construct under discussion [18]. Figure 1 illustrates how learning is perceived as a continuous process that consists of four cycles, namely: experiencing (i.e. interaction with the construct), reflecting (review and evaluate the experience), thinking (drawing conclusions after reflecting on the experience), and acting (apply what has been learned from the process).

Learning can start at any point in the cycle. The benefits associated with experiential learning are: (1) increased opportunities for 'analytical' reflection on tasks completed — in particular 'short term experiential learning' where, similar to our study, the task do not have a long duration [28,38]; (2) 'substantive' benefits which refer to the ability of students to relate theoretical constructs to the practical exposure on a deeper level than just theoretical exposure [38]; (3) 'methodological', which refers to practically applying concepts in a structured way [38]; (4) 'pedagogical', which refers to active participation of the learners in their own and peer learning [38]; (5) 'transition', which refers to bridging the gap between applying concepts during theoretical studies and the practical requirements of industry practitioners. Lee identified lower-level benefits after a comparison between in-classroom learning and field-based experiential learning activities [23], which included an increase in 'soft' skills (e.g. the ability to adopt to change), 'leadership' and 'financial management' skills. Accordingly, learners could establish their own networks of practitioner contacts [4].

3 Education in Data Science

Davenport and Patil describe a data scientist as a "hybrid" of "data hacker, analyst, communicator and trusted adviser" [8] with the abilities to: write

program code, understand contextually the environment in which they function, communicate well in order to convey the message contained in the data to various audiences [8, 25]. These skills can only be acquired through the exposure to real-life scenarios.

Goh and Zhang acknowledged the challenge of exposing students to real-life scenarios when working with data [13]. They referred to current educational efforts to teach students about data analytics as artificial and simplified due to the utilisation of ‘canned’ data (a term used to refer to clean, structured data). They, too, adopted an experiential learning approach whereby they offered students an opportunity to work on a data analytics project in partnership with a large ‘Fortune 500’ company (as a live case study). The objectives of their study were: to investigate the influence of the adoption of an experiential learning approach on the teaching of data analytics; to evaluate students’ perceptions and attitudes towards the experiential learning approach; and to identify challenges associated with their experience when working with big data. Their findings suggested that, although learning outcomes were met and student motivation increased, the students were overwhelmed by the task of statistical analysis of big datasets. Their project entailed additional time challenges as students required more communication with the instructors as well within their groups. Their groups also experienced many failures which—although part of the normal learning process—had to be explained to the (disappointed) students.

Serrano (et al.) adopted experiential learning methods as part of an ongoing data science teaching project focusing on deep learning [9]. An ‘incremental’ teaching approach was used to allow students to adequately reflect their experiences when engaging with the content presented. From the ‘lessons learned’ they proposed the development of a platform for experiential learning that will act as a repository for capturing and storing students’ experiences, to be used by both other students and instructors. They furthermore provided a detailed list of functionalities that such a repository should offer, such as a rating system to gauge the ‘difficulty’ of students’ experiences as well as an anonymous peer review functionality to facilitate students’ reflections.

As part of their investigation of the utilisation of predictive analytics in a supply chain management environment, Schönherr and Speier-Pero evaluated the curriculum of data scientists [29]. They found that in one particular instance, where an experiential learning approach was adopted (in which students had to complete a ‘corporate analytics project’ within an organisation), students were immediately employable by organisations at above-average remuneration (in line with industry requirements).

4 CRISP-DM: The Cross-Industry Standard Process for Data Mining

The CRISP-DM methodology was introduced by a consortium of manufacturing companies, as well as software and hardware organisations in an attempt to standardise and formalise a method for data mining [26]. Their result was

an independent conceptual model which proposed data as central to a six-step process to be followed during a data mining cycle. These six steps started with a clear understanding of the business under investigation, the data to be analysed, the preparation of those data (e.g. cleansing), the application of specific models to analyse the data (e.g. linear regression models), the evaluation of the results as a consequence of modelling, and finally the deployment or distribution of the results and/or the model to the stakeholders.

The CRISP-DM methodology is similar to other data mining approaches such as the Knowledge Discovery Databases (KDD) process model or the Sample, Explore, Modify, Model, Assess (SEMMA) model [30]. Although the numbers of data mining steps differ amongst those (9 in KDD, 6 in CRISP-DM, 5 in SEMMA), the meanings of the steps are similar. For example, the ‘data understanding’ step (2) of the CRISP-DM methodology corresponds to the ‘selection of a data sample set’ and the ‘exploration of the data sample set’ in SEMMA as well as to the ‘selection and preprocessing of data’ in KDD [1].

The CRISP-DM methodology was furthermore chosen for the purpose of this experiential learning exercise for the following reasons: (1) it was conceptual and therefore applicable to any scenario in working towards understanding data; (2) it is a complete and workable methodology [30,37]; (3) the methodology was already prescribed in fourth-year postgraduate studies and therefore seemed appropriate for preparatory purposes also at undergraduate level; (4) the SEMMA method is tightly linked to the SAS enterprise software suite and is thus too software-tool-specific [30].

5 Case Study Description

123 students were enrolled in our third-year second-semester course ‘Trends in Information Systems’. The course offered an introduction to a variety of novel concepts such as IS security and bitcoin technologies (to name a few). As part of the course, students were also exposed to the concept of Big Data and the subsequent challenge of working with unstructured data. The *learning outcomes* specifically for these sessions were:

- *Understand* the Data Lifecycle as part of the Software Development Lifecycle;
- *Describe* the characteristics of unstructured data;
- *Overcome* challenges associated with unstructured data;
- *Understand and implement* the six steps of the CRISP-DM methodology.

At the end of the sessions students were given a practical assignment to complete. The main objective of the practical assignment was to use the current concepts explained during the sessions and apply it in order to work with unstructured data. The task instruction was to follow the CRISP-DM methodology to identify, clean and interpret data from any publicly available, unstructured social media platform (for example Twitter or Facebook) or to select one of the unstructured datasets supplied by the instructors. Students who chose their own datasets were free to use any publicly available source, whereby no further information

was supplied to them (for example about how to use Twitter or what topics or ‘threads’ to use). Students were allowed to use any free tool to assist them in the process of data acquisition, data cleansing, analysis and presentation. To help them on the way, a practical example of what the intended outcome should look like was presented in class.

Table 1. Practical assignment rubrics

CRISP-DM STEP	
1. Business understanding:	What type of business will benefit from this analysis? What are the goals of the business, i.e. what do they want to achieve as a business? What question would you like to answer with the exercise? How will you go about to answer the business question (high level plan)?
2. Data understanding:	Collect unstructured data (from any source). Tip: Twitter might be the easiest. Describe, explore, verify data
3. Data preparation:	ETL: extract, transform (i.e. clean), load data into another structure (flat file, table, etc.). Include copies of your screen where you prepared your data
4. Data modelling:	Decide what you are going to do with the data: apply complex statistical algorithms, or do basic modelling, for example categorisation. Include copies of your screen where you modelled your data
5. Evaluation:	What does the results mean? Do I need to repeat the analysis? Include copies of your screen where you show your results
6. Deployment:	If you were to share your results with the other students: how would you do that? What was your experience working with the datasets (good or bad)? Was the dataset appropriate for what you wanted to achieve? What challenges did you face?
7. Conclusion:	Did you enjoy the assignment? What did you like? What did you dislike? What did you learn?

As part of the assignments the students had to write a report using the six steps of the CRISP-DM methodology. Table 1 outlines the details of each step. Students were also instructed to include copies of the screen(s) where the actual process of data preparation, modelling and evaluation was followed. There was no need to actually deploy or implement their proposed solutions, though the students had to make plausible suggestions about how an organisation could possibly use the results of their analyses.

A conclusion section (7: not originally in CRISP-DM) was added for our students to summarise and reflect what they have learned. Marks were allocated to each of these sections. Table 1 also shows a summary of the rubric used for evaluating the assignments.

6 Method

We followed an *interpretive approach* to analyse a sample of 20 (out of 123) assignment submissions. Saturation was reached with 15 assignments (i.e., no new concepts emerged), but another 5 assignments were analysed to confirm saturation. All data sets used by these students were publicly available and not password-protected. Although the individual data records used by the students did not disclose any identifying attributes, the organisations associated with these data sets were in some instances revealed. We followed the ethical procedure recommended by Langer and Beckman who suggested that if public data, that is not password-protected, is used, researchers do not need to obtain any usage permission [22]. Anyway, the anonymity and privacy of the users were guaranteed as we anonymised particular organisation names by grouping them into ‘industries’.¹ All assignments followed the structure of the rubrics of Table 1.

Thematic content analysis was used to analyse the students’ project reports.² Thereby we followed the six steps proposed by Braun and Clarke [6]. Initial *codes* were captured as they emerged and were recorded under every step of the CRISP-DM methodology. As the analysis continued and themes emerged (and were reviewed and named), it was easy to see how the *six steps of the CRISP-DM methodology mapped to the four stages of the experiential learning approach*. The following section presents the analysis and discussion of these findings.

7 Analysis and Discussion

The analysis and discussion outline section followed the phases of the experiential learning approach, namely: abstract conceptualisation, concrete experience, reflective observation, and the lecturers’ reflection. Each of the six CRISP-DM steps could be related to the four phases of the experiential learning approach.

7.1 Abstract Conceptualisation: Business Understanding

The abstract conceptualisation stage is concerned with learners trying to make sense of a problem at hand. For our particular assignment the students could use any publicly available data set from social media or a variety of unstructured datasets supplied to them, (see above). The objective was to understand the message(s) the data can communicate to various audiences, and questions that can be formulated which can be answered with the help of the gathered data. Most students chose data from Twitter from diverse industries namely: Gaming, Finance, Music, Government/Activism, Government/Treasury, App Store, Fast-moving consumer goods (Beverage), Marketing/Communications.

¹ For this paper we obtained the consent from our participating students as well as a permission from our faculty’s ethics committee.

² The 1st author (S.E.) was the examiner of the assignment and was thus familiar with the content of the assignment. The 2nd author (M.J.H.) familiarised herself with the content by reading two assignments before the analysis began.

It turned out that our students were able to contextualise those data and could identify the parties that might be interested in answers to questions that relate to the data. For example, Participant #5: *“These businesses want to know their market/customers better while situating themselves to a favourable position in their operating markets”*. Contextualising results is a very important skill of any data scientist [8, 17, 25]. Also Kennan stated that in a business environment it is important for a data scientist to know what the organization does, who its customers are, and what the operating environment is [17]. She further stated that the context is different for different countries due to different governments’ regulations. Hence it is not required for graduates to know all the contextual details; nonetheless they must be aware that *“the contexts in which data and information are used are highly varied”*; they must also *“understand examples, and where to look for specific contexts, and be prepared to continue learning on the job”* [17].

In order to contextualise their potential results, the students were required (as part of the CRISP-DM methodology) to find out more about the companies, understand their ‘missions’, ‘visions’ and goals. This aspect of the assignment was very important, as the students were exposed to real-life companies and had to make sense of how a data set supports an organisation’s purposes. A few students did this very well by studying a business, understanding its goals and how the social media data relate to those goals. This exposure to real-life scenarios is important in delivering industry-ready graduates [13, 32]. For example, participant #3 noted: *“it is very important that the academy has a strong social media presence to attract potential donors, spread the word about the music programs on offer, and to promote any upcoming events”*. Accordingly, Kennan states that in order to understand data within its context one has to understand the intended audience of the information [17].

7.2 Active Experimentation: Data Understanding and Preparation

The active experimentation stage is concerned with planning the *“forthcoming experience”* [15]. In this instance the students had to plan, in accordance with the CRISP-DM methodology, how to approach the collecting, extracting, cleaning, loading and storing of their chosen data sets. Part of this Extract-Transform-Load (ETL) process was the verification of the sources.

The students used a number of techniques to clean the data sets, such as splitting datasets into smaller parts according to the original date into day, month and year, then combining those attributes into a new column. The columns were furthermore labelled with meaningful names, classified according to types of Tweets (for example RT for retweet, OR for original tweet), and classified according to keywords. The ‘noise’ of hyperlinks was removed. All of these activities are essential for their ability to work with data [8, 25].

Furthermore, students understood the importance of recognising missing data and the potential implications it might have on the analysis results—or that it might not affect the result depending on the way the data is analysed. Through the ETL process the students were also able to identify and discard redundant

Twitter Analytics, Zoho Reports (now Zoho Analytics), Tableau, MS Excel Azure Machine Learning add-in, and Jupyter Notebook in Python. The variety of tools available to students indicated the evolving nature of data science. Accordingly, exposure to a variety of such tools will increase the students' abilities.

7.3 Concrete Experience: Data Modelling, Evaluation and Deployment

The concrete experience stage is concerned with the actual completion of the activity. During this assignment, this stage refers to the modelling, evaluation and deployment of the results.

During this stage we observed that the students used a variety of *visualisation* methods to communicate their results, for example: bar charts, pie charts, scatter plot, bubble chart, ring graph, line chart, and location map. One student used a histogram to indicate how brand sentiment changed over a period of time. Some students used more than one visualisation method to communicate different messages. One student used a more advance modelling technique, namely the 'predict' and 'simulate' functions of RapidMiner, to build a Deep Learning Simulator based on the data set. Accordingly, Kennan stated that there is a great need for graduates to have such visualisation skills [17], which allow them to present the data in such a way that it enables managers to make better and quicker decisions and to communicate messages more clearly to stakeholders outside an organisation.

A second observation at this stage was the students' ability to interpret the visualised results. The majority of the students were able to correctly interpret the meaning behind the visualisation. The power of big data lies indeed in the *interpretation* of analysis results. McAfee and Brynjolfsson stated accordingly: "*Big data's power does not erase the need for vision or human insight*" [25]. Our assignment was not too complex, and some of the analyses were quite basic, but nonetheless contained powerful messages. For example, participant #3 found that "*the company does not have to change what it is tweeting; rather when it is tweeting. In addition, gaining more followers should increase impressions and engagement rate. If they do these two things, they should see an improved Twitter performance*". Students were thus able to derive *meaning* from the data, such as: "*the rebranding campaign was not well received*" (based on the tweets analysed) "*as the audiences did not understand the campaign concept when it came to the brand messaging and intent. Some found it offensive and insensitive whilst others either engaged positively or were indifferent*" to what the company had communicated.

An important component of the methodology was to evaluate the results to verify if they were plausible. This requires students to critically double-check their initial findings. Whilst most students reported that their results were in line with their expectations, participant #12 found that a sentiment analysis done by Azure were not correct, as sometimes there was a colloquial misunderstanding which skewed the results. Participant #7 evaluated the data post modelling and concluded that the results were not accurate because re-tweets

skewed the analysis. This illustrates a level of awareness about the nature of the data set which is very important for any data scientist. Accordingly, Costa and Santos describe data scientists as having an ‘inquisitive mind’ with which they ‘interrogate’ the data to understand their deeper meaning [7].

Finally, students recognised that the *dataset can potentially answer different questions depending on the analysis*. For example, participant #7 generated nine different visualisations from the dataset which included a pie chart, five different bar charts, a scatter plot and two line charts. The scatter plot was used to indicate location. Participant #8 developed a generalised linear model between the categories. However, the fact that one data set can communicate different messages is something students struggle with. Our assignment, that prescribed the adopted CRISP-DM methodology, allowed students to do an arbitrary number of modelling iterations whereby a result obtained after the completion of one cycle of the methodology introduced a new question or problem to be investigated.

7.4 Reflective Observation

The reflective observation stage is concerned with the students’ reflection on their completed activity. This stage offered students the opportunity to indicate what aspects of the assignment as well as experiential approach they enjoyed and what they found challenging. From the lecturer’s perspective, the reflective observation stage allowed her to make a judgement about the success of the exercise, (see Subsect. 7.5). Some positive feedback by the students regarding the assignment included the following points:

- Students enjoyed the ‘mining’ aspect of the assignment. This refers to practically using an identified software tool, (see Subsect. 7.2). Accordingly, Kolb explained that students preferring a practical approach to solving problems refer to the ‘converging’ learning style [18].
- The practical component of the assignment also extended to learning new software. Participant #10 stated: “Overall, I enjoyed working on the assignment because I enjoy working with new software, and the learning that comes with it, especially when it allows you to apply your theory, making it interactive”. Learning how new software works whilst completing an assignment is an example of ‘incidental’ learning which is imperative in assisting students to get ‘hands-on’ experience and to prepare them for the information age. Incidental learning is a ‘side effect’ of learning whereby the students were initially not aware of the fact that they would have to learn new software, but then suddenly had to acquire new skills in order to complete the entire CRISP-DM lifecycle [31].
- Students learnt about the potential impact of data. Participant #4 said that he could “*see the potential impact of big data*” whereby “*it was best to see this when it was done practically*”. Participant #2 confirmed this by stating: “*I enjoyed seeing how the steps are done practically and I learnt a great deal about how unstructured data can be used to make better business decisions*”. Participant #1 indicated that “*simple data can give good answers to important questions*”.

- Students learnt about a variety of options to visualise data as well as the power of tools to manipulate data. Participant #7 stated: “I was amazed how these tools could formulate meaningful graphs and charts based on unstructured data. Even if a field was null, the tool was able to identify it without mixing it with the rest of the data”. Accordingly, Wang (et al.) stated that the adoption of good visualisation methods can transform the challenges introduced by big data (such as the vast volumes of seemingly unrelated data) into meaningful ‘pictures’ [35].
- The assignment allowed for Self-Motivated Incremental Learning (SMIL) [5]. SMIL is concerned with the intrinsic motivation by a person which will allow him/her to learn a hierarchy of skills freely by repeating three phases: exploring the environment, identifying interesting situations, and obtaining skills to cope with these situations. Participant #15 reported that *“we were never really taught how to actually do a full data analysis. We were only ever shown the theory behind data analysis and data modelling, and we had to use the information along with all the other knowledge we have from Statistics, Mathematics, IT, etc., and figure out ourselves how to work with data and analyse the raw data. I learned that Excel is a much more powerful tool than I first anticipated, but it cannot compete with how strong Python is and how easily you can achieve the same results”*. The assignment furthermore introduced students to the area of data science, whereby one participant noted: *“It sparked an interest in Python in me and I already enrolled in two short online courses on Python and R in data analytics”*.

Some challenges observed by the students included the following points:

- Data preparation took a long time due to the volume of data they had to work with. The majority of the students indicated that this was their least favourite part of the assignment. Participant #10 indicated that *“having to read through the data, and generating a question or problem statement, took a while”*.
- Some students struggled to understand the data set. Participant #3 said that it was a *“challenge to understand what data I was working with, and the relevance it might have”* to the company in question. *However, after doing plenty of research I was able to overcome this*. This case is another example of SMIL [5], as the student has to analyse the environment in order to solve the data problem.
- One student indicated that he would have preferred more direction in the assignment, and *“often felt lost and unsure of whether I was doing things correctly”*.
- A few students reported that it was difficult to identify the correct technology for the task. Participant #7 stated that *“finding a data analyser tool which would best fit the dataset was challenging”*. Participant #20 stated that *“being able to choose the right model, and making the data fit the model, was also a challenge”*. Participant #16 struggled with the method: *“I did not know which algorithms to use and how to correctly use the datasets”*. Obviously this challenge was two-fold:

- firstly w.r.t. the students' ability to identify the technology with the correct functionality to obtain the envisaged results,
 - secondly w.r.t. the students' ability to use the chosen software (as some software requires deeper knowledge for its proper utilisation).
- Finally, some students also had problems to identify an appropriate visualisation method.

The above-mentioned challenges are linked to key competencies of data scientists [7]. Exposing the students early to these challenges should give them an opportunity to continue with SMIL in preparation for a workplace after graduation.

7.5 The Lecturer's Reflection

This subsection presents the reflection of the lecturer during and after the assignment. We consider (1) the appropriateness of the prescribed methodology, (2) the students' performance, and (3) 'lessons learnt'. Each of these will be briefly discussed.

Appropriateness of the CRISP-DM Methodology. Prescribing the CRISP-DM methodology was appropriate, as students easily grasped the six steps of the process while working with data. The methodology guided students through the process and provided a structure for approaching an assignment that seemed daunting to some students at first. Due to its conceptual nature, students can hopefully reuse this methodology when working on similar projects in the future.

Students' Performance. 11% of our 123 students obtained an assignment mark between 80% to 89%. About half of all students obtained a mark between 70% to 79%. About a quarter of the students obtained a mark between 60% to 69%. The remaining students obtained a mark between 50% to 59%. The average mark for the assignment was 68%, the highest 88%. The students who obtained a mark between 50% and 59% did either not provide enough detail in their reports, or misinterpreted their found results. For example, two students discovered the most prominent words associated with their data but failed to synthesise the findings to draw a meaningful conclusion. As a consequence, they did not adequately answer the initial question. Overall, given the performance of the students, the assignment was successful in teaching students the basic skills necessary to work with unstructured data.

Lessons Learnt, and Recommendations. Our experience can be summarised as follows.

- *Working with social media data* was enjoyed by our students, as they are already familiar with these platforms. On the basis of this experience we recommended that educators use data sets that students can relate to (such as social media).

- Students found the extracting, cleaning and transforming of data very challenging and time-consuming. This was their biggest and to some extent overwhelming tasks. Therefore we recommended that students be provided with more practical demonstrations of how to extract, clean and transform data, as the ETL process is the biggest task when working with any form of data.
- The lecturer should cater for and accommodate students with different learning styles. Our assignment was best suited for students with the ‘converging’ learning style (who prefer to solve problems and apply their knowledge to practical implementations) as well as to students with the ‘accommodating’ learning style (who prefer to ‘do things’ practically) [18]. Students with the ‘diverging’ style of learning (who prefer to watch rather than do) as well as with the ‘assimilating’ style (strong analysts when given good information) should be accommodated by group-work opportunities.
- The lecturer should offer students more consultation time when students have more questions particularly at the beginning of the assignment; for comparison see Goh and Zhang [13].

8 Conclusion

This paper describes the effectiveness of teaching basic skills to third year undergraduate students to work with unstructured data by following an experiential learning approach (ELA). We found that the ELA enabled ‘novices’ to acquire basic skills in working with unstructured data. The students were exposed to a structured methodology that allowed them to tie the data sets to the goals of a business. They used a variety of tools and technologies to obtain, prepare, model and interpret unstructured data sets. Our assignment enabled the students to experience the ‘nature of data’, the influence of missing or redundant items on the analysis results, and how one data set can provide different answers to a variety of questions. The students reported that they enjoyed the ‘(data) mining’. The students reported that they also enjoyed the acquisition of skills to work with new software and tools. They realised the impact social media data has on an organisation. The challenges experienced by the students in completing this assignment do not outweigh the benefits that students derived from it. As more organisations become data-driven, graduates need to be prepared to assist organisations in their data needs [3]. With our educational efforts we also gave our undergraduate students an initial preparation for possible postgraduate studies [24] in this new field.

References



1. Azevedo, A., Santos, M.F.: KDD, SEMMA and CRISP-DM: a parallel overview. In: Proceedings European Conference on Data Mining, p. 6 (2008)
2. Baškarada, S., Koronios, A.: Unicorn data scientist: the rarest of breeds. Program 51(1), 65–74 (2017)

3. van Belle, J.P., Scholtz, B., Njenga, K., Serenko, A., Palvia, P.: Top IT issues for employers of South African graduates. *CCIS* **963**, 108–123 (2019)
4. Bobbitt, L.M., Inks, S.A., Kemp, K.J., Mayo, D.T.: Integrating marketing courses to enhance team-based experiential learning. *J. Mark. Educ.* **22**(1), 15–24 (2000)
5. Bonarini, A., Lazaric, A., Restelli, M.: Incremental skill acquisition for self-motivated learning animats. In: Nolfi, S., Baldassarre, G., Calabretta, R., Hallam, J.C.T., Marocco, D., Meyer, J.-A., Miglino, O., Parisi, D. (eds.) *SAB 2006. LNCS (LNAI)*, vol. 4095, pp. 357–368. Springer, Heidelberg (2006). https://doi.org/10.1007/11840541_30
6. Braun, V., Clarke, V.: Using thematic analysis in psychology. *Qual. Res. Psychol.* **3**(2), 77–101 (2006)
7. Costa, C., Santos, M.Y.: The data scientist profile and its representativeness in the European e-Competence framework and the skills framework for the information age. *Int. J. Inf. Manage.* **37**(6), 726–734 (2017)
8. Davenport, T.H., Patil, D.J.: The Sexiest Job of the 21st Century. *Harvard Business Rev.* (October), 8 (2012)
9. Emilio, S., Martin, M., Daniel, M., Luis, B.: Experiential learning in data science: from the dataset repository to the platform of experiences. In: *Ambient Intelligence and Smart Environments*, pp. 122–130 (2017)
10. Eybers, S., Hattingh, M.J.: The last straw: teaching project team dynamics to third-year students. *CCIS* **963**, 237–252 (2019)
11. Fosso Wamba, S., Akter, S., Edwards, A., Chopin, G., Gnanzou, D.: How big data can make big impact: findings from a systematic review and a longitudinal case study. *Int. J. Prod. Econ.* **165**, 234–246 (2015)
12. Gantz, J., Reinsel, D.: *Extracting Value from Chaos*. IDC, p. 12 (2011)
13. Goh, S., Zhang, X.: Incorporating experiential learning into big data analytic classes. In: *Proceedings 21st Americas Conference on Information System*, p. 10 (2015)
14. Grace, S., Innes, E., Patton, N., Stockhausen, L.: Ethical experiential learning in medical, nursing and allied health education: a narrative review. *Nurse Educ. Today* **51**, 23–33 (2017)
15. Healey, M., Jenkins, A.: Kolb's experiential learning theory and its application in geography in higher education. *J. Geogr.* **99**(5), 185–195 (2000)
16. Janvrin, D.J., Watson, M.W.: Big data: a new twist to accounting. *J. Account. Educ.* **38**, 3–8 (2017)
17. Kennan, M.A.: In the eye of the beholder: knowledge and skills requirements for data professionals. *Inf. Res.* **22**(4), 1–21 (2017)
18. Kolb, D.A.: *Experiential Learning: Experience as the Source of Learning and Development*. Prentice Hall, New Jersey (1984)
19. Kolb, Y.A., Kolb, D.A.: Learning styles and learning spaces: enhancing experiential learning in higher education. *Acad. Manage. Learn. Educ.* **4**(2), 193–212 (2005)
20. Kotzé, E.: A survey of data scientists in South Africa. *CCIS* **730**, 175–191 (2017)
21. Kotzé, E.: Augmenting a data warehousing curriculum with emerging big data technologies. *CCIS* **730**, 128–143 (2017)
22. Langer, R., Beckman, S.C.: *Sensitive Research Topics: Netnography Revisited* (2005)
23. Lee, S.A.: Increasing student learning: a comparison of students' perceptions of learning in the classroom environment and their industry-based experiential learning assignments. *J. Teach. Travel & Tourism* **7**(4), 37–54 (2008)

24. Marshall, L., Eloff, J.H.P.: Towards an interdisciplinary master's degree programme in big data and data science: a South African perspective. *CCIS* **642**, 131–139 (2016)
25. McAfee, A., Brynjolfsson, E.: *Big Data: The Management Revolution*, vol. 9. Harvard Business Review, Brighton (2012)
26. North, M.: *Data Mining for the Masses*. CreateSpace Independent Publication Platform, Scotts Valley (2016)
27. Piatetsky, G.: How many Data Scientists are there and is there a Shortage? Technical Report, (2019). <https://www.kdnuggets.com/2018/09/how-many-data-scientists-are-there.html>
28. Scarce, R.: Field trips as short-term experiential education. *Teach. Sociol.* **25**(3), 219 (1997)
29. Schöherr, T., Speier-Pero, C.: Data science, predictive analytics, and big data in supply chain management: current state and future potential. *J. Bus. Logistics* **36**(1), 120–132 (2015)
30. Shafique, U., Qaiser, H.: A Comparative study of data mining process models (KDD, CRISP-DM and SEMMA). *Int. J. Innov. Sci. Res.* **12**(1), 217–222 (2014)
31. Sleight, D.: *Incidental Learning from Computerized Job Aids*. Technical Report (1994). <https://msu.edu/~sleightd/inclearn.html>
32. Smuts, H., Hattingh, M.J.: Towards a knowledge conversion model enabling programme design in higher education for shaping industry-ready graduates. *CCIS* **963**, 124–139 (2019)
33. Tanwar, M., Duggal, R., Khatri, S.: Unravelling Unstructured Data: A Wealth of Information in Big Data. In: *Proceedings ICRITO 4th International Conference on Reliability, Infocom Technologies and Optimization*, pp. 1–6, IEEE (2015)
34. Tole, A.: Big Data Challenges. *Database Syst. J.* **IV**(3), 31–40 (2013)
35. Wang, L., Wang, G., Alexander, C.A.: Big data and visualization: methods. *Challenges Technol. Progress. Dig. Techn.* **1**(1), 33–38 (2015)
36. Wang, S., Yeoh, W., Richards, G., Wong, S.F., Chang, Y.: Harnessing business analytics value through organizational absorptive capacity. *Inf. Manage.* **56**, 103–152 (2019)
37. Wirth, R., Hipp, J.: CRISP-DM: towards a standard process model for data mining. In: *Proceedings 4th International Conference on the Practice Application of Knowledge Discovery and Data Mining*, pp. 29–39 (2000)
38. Wright, M.C.: getting more out of less: the benefits of short-term experiential learning in undergraduate sociology courses. *Teach. Sociol.* **28**(2), 116 (2000)



Connecting Generation Z Information Systems Students to Technology Through the Task-Technology Fit Theory

Adriana A. Steyn^(✉), Carina de Villiers, Joyce Jordaan,
and Tshegofatso Pitso

Department of Informatics, University of Pretoria, Pretoria, South Africa
riana.steyn@up.ac.za

Abstract. This study investigated how an interactive e-resource could be used to increase students' performance for a specific Information Systems assignment given. As academics we are struggling to find sources that really talk to 'Generation Z' in the way they prefer to learn. We wanted to determine if we can create such a resource to increase students' performance. This study investigates the usefulness of a self-created e-textbook for Systems Analysis and Design through the task-technology fit theory lens. A quantitative data analysis was conducted on a group of undergraduate Information Systems students. A significant association between the characteristics of the tasks and the technology used to perform the specific task was found. A significant association between the students' understanding of the work and improving their knowledge as well as their contributions to a team was also found. Generation Z relies heavily on peers for assistance even though literature says that their social skills are under-developed. As academics we need to understand the Generation Z, and how they prefer to study, and then create content and tools for them so that they can broaden their knowledge and become life-long learners. Higher education institutions should become more student-centered and less lecturer-centered.

Keywords: E-textbook · E-resource · Interactive textbook · Generation Z · Millennials · Task-technology fit theory · Information systems education

1 Introduction

Throughout the years, many authors have tried to answer the question: how do students learn [12, 17, 18]? Already in 1987, Chickering and Gamson wrote a paper entitled *Seven principles for good practice in undergraduate education* [6]. They acknowledged that there was a problem in undergraduate teaching and emphasised the importance of having commitment from faculty members

and students. Their seven principles are: (1) encouraging contact between students and faculty members, (2) developing reciprocity and cooperation among students, (3) encouraging active learning, (4) giving prompt feedback, (5) emphasising time on task, (6) communicating high expectations, (7) respecting diverse talents and ways of learning. Though [6] was already published in 1987, the same question is still asked and the problem is still relevant [20].

One of the mechanisms identified to adapt the ‘old’ education system is ‘technology’, a tool in which we can engage more with students as they are exposed to and used to technology from a fairly young age [19]. They are almost ‘born with a phone in the hand’ [17]. This is the generation sitting on our campuses today. Gone are the ‘millennials’; now we are engaging *Generation Z* [14, 15, 20]. And yet it is believed that the education system caters for the ‘old’ generation of millennials, and even prior to them, and not necessarily for Generation Z [14], because these students are changing annually. However not a lot of evidence shows that the technology we use on a daily basis can even be used for education and learning as students need to be engaged in the learning process [3]. New students enter our campuses annually, which makes adaption of our teaching approaches difficult. Technology, too, is changing so rapidly that one can hardly keep up. Even the ‘powerful’ PowerPoint presentations are already considered outdated [12, 15]. We need to find the best-fit technology for the specific task at hand and see how it works and hopefully that it works. Several publications recommend that more visual tools should be explored as they proved to enhance the learning experience and make students more excited about their studies [12, 15, 17], such as YouTube, infographics, colorful images, and the like. Shatto and Erwin went so far as to say that one should limit reading to only relevant information [15]. How students use the textbook and its features as well as the instructor’s usage should be investigated to see if there is a possible link between the two [7].

This paper explores the notion of a *lecturer-designed interactive e-resource*—some call it an ‘interactive textbook’—and how students used the textbook to carry out a specific task. Thereby we followed the *task-technology fit theory*. In this paper we will call it an ‘interactive textbook’. Our aim is to investigate the usefulness of the resource specifically for our *Systems Analysis and Design* module. Thus this paper suggests that there is a positive association between the interactive textbook and the actual task which the students had to do.

2 Background

2.1 Millennials Versus Generation Z

Although many authors differ as to when generation Z was born and who should be included, it seems as if they agreed on individuals *born from 1995 onwards* [4, 5, 8, 9, 13–15]. Monaco and Martin’s study can already be regarded as ‘old’ as they still talk about the ‘millennials’ [12], but they make some interesting arguments about how students learn and, more specifically, their characteristics. They list seven general characteristics, most of which correlate with [6].

These are: (1) they feel ‘special’—*we are all winners just by participating*; (2) they feel sheltered—*baby on board* signs, parent-driven schedules, little free-time, hence not much free thought on daily planning (limitation for educators); (3) they are team-oriented and less comfortable working alone; (4) they are confident and highly optimistic, with instant access to information at any time, and modest commitment to homework; (5) They are or feel pressured, which leads to a longing for ‘instant feedback’; (6) They have a strong desire to ‘achieve’; (7) They are ‘conventional’ again with a new respect for ‘culture’.

According to [5], by contrast, generation Z is connected and craving for a digital world, but their social skills are underdeveloped and they do not feel safe, which is strongly different from the millennials. They are more individualistic and have an increased risk of isolation, anxiety and depression. But they also want feedback immediately and conveniently [15]. They are also more accepting of and open-minded about difference [15]. This different picture should be considered by educators, as we cannot assume the same character traits of millennials and think we are still ‘engaging’ our students. These changes in the students’ mindsets are forcing higher education institutions to become more student-centered and less lecturer-centered.

As academics, according to [12], we need to take a step back, out of the so called ‘lime-light’, understand the students entering our gates, and ask them how they prefer to learn and what they want to see [12], because our education system was never designed with them in mind.

Shatto and Erwin as well as Vikhrova note that as educators, we have to understand that generation Z see their technology and gadgets as integral to their lives and that they actively use technology in all spheres of their lives [15, 20]. Therefore they are also multitaskers, but not the way we think they are. They have the ability to skip quickly between tasks, even if the activities are unrelated to one another. Generation Z wants to learn by observing, with practical applications [15], in a more ‘hands-on’ approach [14]. These students also prefer to learn independently on their own [14]. They see peers and educators as valuable ‘resources’, but they will engage on their own terms. And lastly, Vikhrova stated that they are ‘clip-thinkers’ [20]—in other words: they view fragments of images, facts, videos, and process these as a whole so that they can form the big picture. It is noted that clip-thinking helps the brain from congestion and thus acts almost as a filter of information.

Seemiller and Grace noted that there are four things campuses can do to engage with generation Z students [14]: (1) utilize video-based learning; (2) incorporate intrapersonal learning into class and group work, thereby breaking a bigger project into smaller manageable sections; (3) offer community engagement opportunities; (4) connect generation Z students to internship opportunities. Of these four approaches only the first two (1–2) will be considered in this paper.

2.2 E-Textbooks

Key aspects of an e-textbooks are its ‘mobility’ [3] and how it can ‘carry’ more resources than a traditional book. Due to these features, educators can create



Fig. 1. E-textbook design example



Fig. 2. Task-technology fit theory according to [10]

more customized interactive textbooks [3]. This allows the creator of such textbooks to focus more on their contexts of delivery. Bikowski and Casal acknowledged that a large amount of research has already gone into textbook design—however: *“little has been done on customised, interactive textbooks designed within a specific content and with specific course outcomes in mind”* [3]. This paper aims to change this. When investigating e-textbook affordability for students, Baek and Monaghan stated that the textbook must be of a high quality and must also be easy to use [2]. Our interactive textbook, which we describe in this paper, was designed using a tablet ‘look and feel’, such that its usability appeared familiar to our students. We also ensured that the design was ‘clean’ for the sake of a better quality textbook, (see Fig. 1).

2.3 Task-Technology Fit Theory

According to Goodhue and Thompson one of the strongest indicators for individuals to use technology is if there is a ‘system/work fit’ [10], i.e.: what I want to use the system for will determine whether I will use it. Giving the specific textbook to the students to perform a specific task gives us a plausible reason for applying the task-technology fit theory. This theory states that a user should willingly use the technology for a specific task before we can say that it was ‘effective’ [1,10], (see Fig. 2)

3 Method

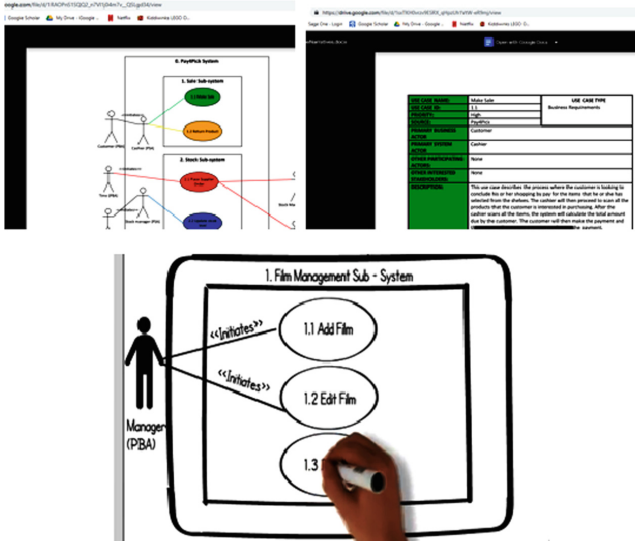


Fig. 4. E-textbook's practical section

One of the key problems of an undergraduate module in our context is the absence of a proper textbook that focuses on all the aspects of the module of interest. This module is a first-year systems analysis and design course with 340 participating students. The students enter the university assuming there is a specific textbook for each module.

The thought of creating our own e-book emerged, and the starting point of the textbook was by getting the *students to contribute to its contents* [16]. The textbook would be cost-effective to develop, and module-specific. The first version of the textbook was launched in July 2018.

After the students completed a specific assignment, where they had to utilise our textbook, they were asked to complete an online survey. No marks were allocated for completion of the survey. The survey data were exported to Microsoft Excel and statistically analysed with IBM Statistics SPSS tool (version 25).

The methods used during the analysis of the data are: frequency analysis per question; multiple response frequency analysis; descriptive statistics such as median, and standard deviation; cross-sectional analysis; graphical analysis such as pie charts and bar charts.

4 Findings and Discussion

4.1 Participants

The total number of responses received from the survey was 171, of which 170 were completed in full. However, as this paper's focus is specifically on generation

Z, it is important to look at the age of the respondents. As this study took place in 2018, and literature stated that generation Z students were born more or less from 1995 onwards, these students should now be at most 23 years old. However, we did not force our students to disclose their date of birth. Only 142 participants answered this question; they fell indeed into generation Z's birth date range. Upon closer analysis we also found that a few students entered '2018' as their date of birth. These useless answers were discarded, such that only 117 usable responses were obtained. All in all our survey had a 34% response rate.

Accordingly, the average age of the students is 20.5 years (mean), with the majority of responses from participants who are 19 years of age (mode). The majority of the students were born in 1998 (35) and 1999 (50) thus correlating with the mean.

Looking at the degrees for which the students study, the majority (76.8%) studies either *BIS Information Science* (14,5%), *BIT Information Technology* (18,8%), or *BCom Informatics Information Systems* (43,5%). Fewer students studied *BSc Information Technology Information & Knowledge Systems* (11,1%) or *BCom Financial Science* (1,7%). *BCom General*, *BCom Statistics*, *BEd FET General*, and *BSc Computer Science* students together were 3% of the respondents, and *BSc Geoinformatics* 7%.

As the e-textbook was made available through our university's online learning management system, students could download it to their devices; some of the files however were located on Google Drive, (Fig. 4). Thus internet connection had to be queried. Only 8 students indicated that they had no internet connection at all at home. However, all students indicated that they have Wifi on campus; thus it seems that there was no internet access barrier to using the e-textbook.

W.r.t. their learning styles and preferences, the students had to indicate how they prefer to learn and who they will go to first for assistance. As shown in Fig. 5, most students prefer case studies; this makes sense as this part of the assignment was practical modelling, and by actually doing it one will learn better. Attending tutor sessions was also a popular learning style, as well as collaboration with fellow students.

In connection with the learning styles, when we asked the students whom they approached first when in need of help, 'group members' was the most frequent answer; (indeed the assignment was a group assignment). This correlates with [14] who stated that students are independent workers but will engage with their 'resources'—fellow students, Youtube or a lecturer—on their own terms.

W.r.t. the 'call' in the literature to make academic tools 'more visual', also in our case most of the answers indicated that the students prefer visual aids. Although our interactive textbook ranked last (Table 1), we hope that this was mainly due to the fact that this was the first time these students were exposed to such a device. As shown in Fig. 6, however, the students used our interactive textbook quite regularly although they previously indicated that they did not use it as *first* point of reference. Hence it seems that our provision was at least somewhat helpful to them.

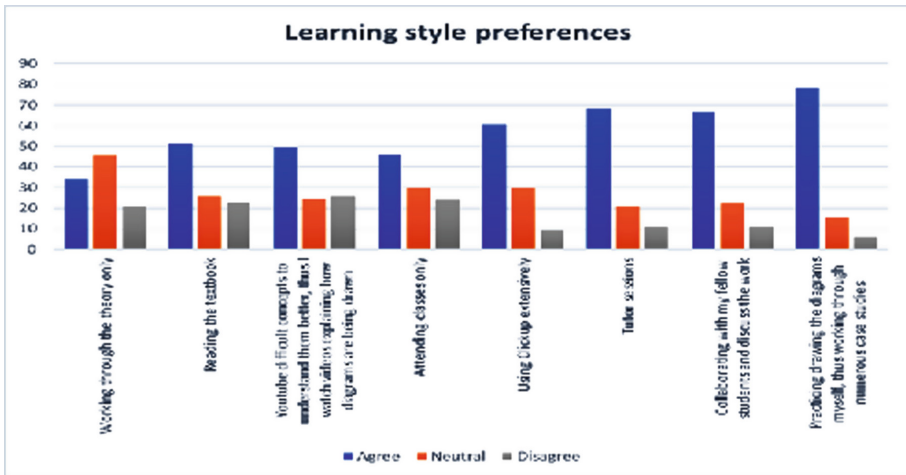


Fig. 5. Learning style preferences

Table 1. Who do you ask first for assistance?

RANK	SOURCE
1	Project group members
2	Prescribed textbook
3	YouTube
4	Assistant lecturers
5	Main lecturer
6	Other students
7	Library
8	Interactive e-textbook

4.2 E-Textbook-Specific Characteristics and Usefulness

As one of the main purposes of our interactive textbook is to provide students with more options to gain knowledge, students were asked to tell: *If I were given practical examples in an electronic format, I would rather study using...* Figure 7 shows that students still prefer classroom interaction with their lecturer, but most of them also like electronic examples. Half of the students said that they prefer the textbook. The rest were rather neutral with only 10% stating they do not prefer using the tool.

One has to understand the students' experience of using the textbook for the specific assignment, based on the various diagrams required to complete the assignment task. Figure 8 shows that students felt that our e-textbook was easily usable for both the UML use case diagrams and the theory sections. They also felt that the e-textbook provided a holistic view of systems. However, most students felt neutral towards this question. Understanding the scope of the system as well

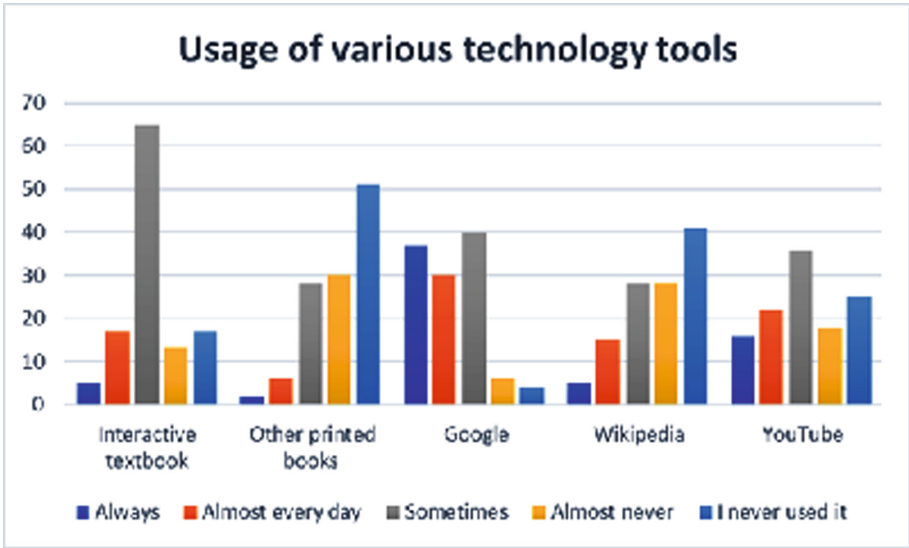


Fig. 6. Usage of various tools for completing assignment

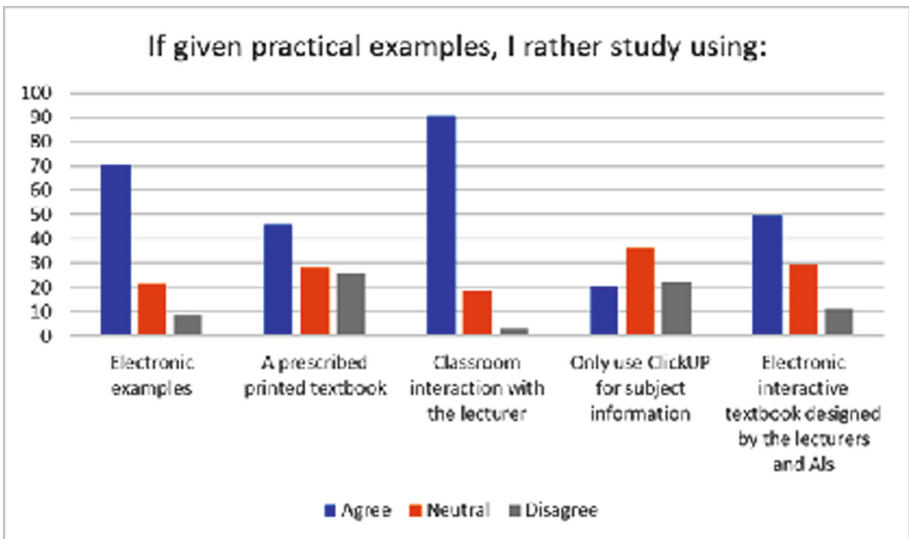


Fig. 7. I would rather study using...

as the e-textbook itself were ‘easy’. What is clear from the results in Fig. 8 is that very few students (no response more than 10%) did not like the textbook in terms of the use case diagrams, narratives, creating a holistic view of systems, understanding the scope of the system or the textbook itself.

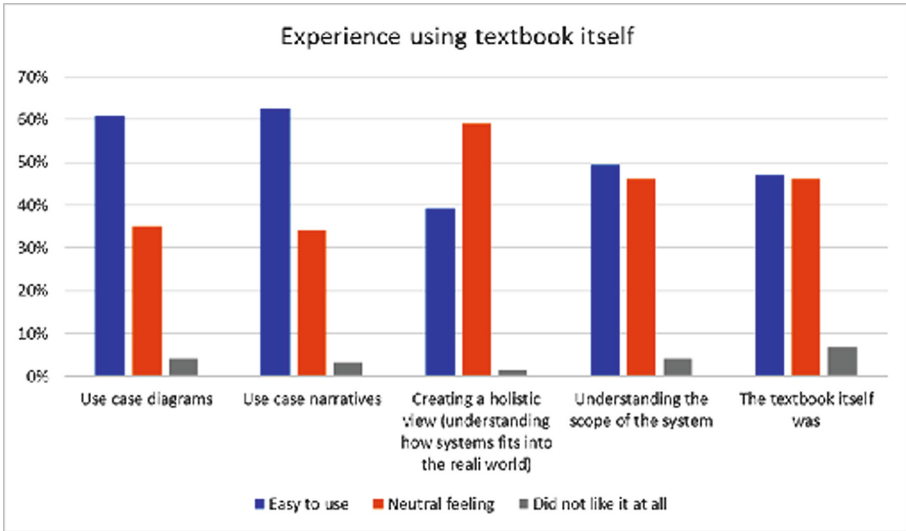


Fig. 8. Experience using the e-textbook

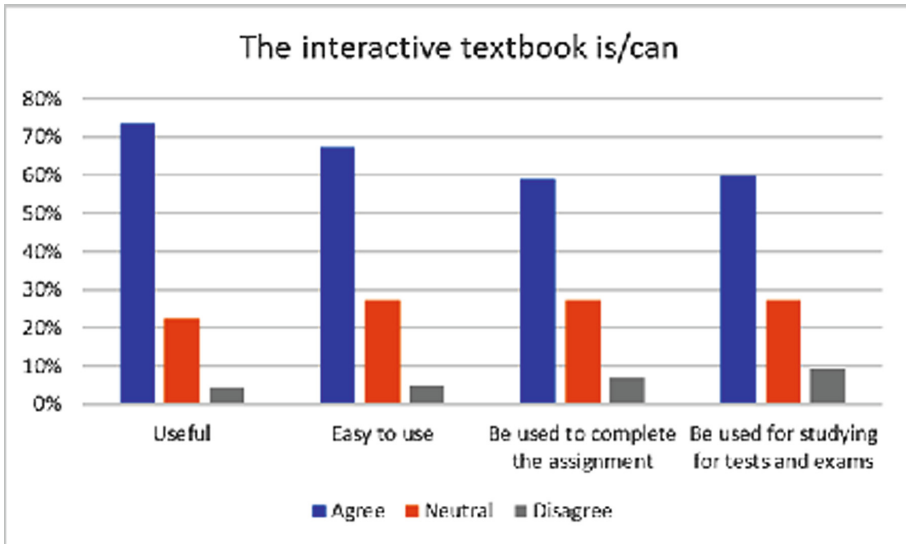


Fig. 9. The interactive e-textbook is/can...

Continuing on the previous questions, students were also asked how they experienced the e-textbook as a whole. All in all they felt the e-textbook was useful, easy to use, and also helpful for the completion their assignment task. They also indicated that it could be used in preparations for tests and exams, as shown in Fig. 9. Few respondents disagree with its usefulness, ease of use, or whether they could use it in the future.

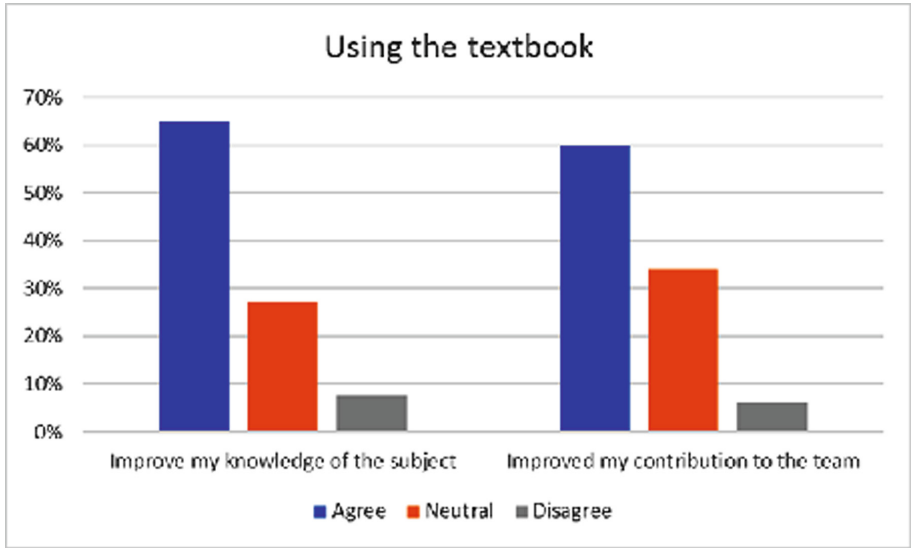


Fig. 10. Student's feelings towards using textbook

4.3 Task-Technology Fit Theory

As mentioned above, the idea behind the task-technology fit theory is to see if using the technology for a specific task did indeed increase our students' performance: see Fig. 10. To determine if our interactive textbook is really linked with actual usefulness and students' performance, a cross-tabulation analysis was done: see Table 2. The standardised residual—if it is 2 (or higher) or -2 (or lower)—is an indication of which cell in the table contributes most to the corresponding χ^2 value. For all the cross tabulation analysis, there was a significant association between the corresponding statements: see the following *interpretations* (§1–§8) for further explanations related to Table 2.

§1 There is a significant association between use case diagrams in the interactive textbook and improving the knowledge of the subject. Standardized Residual is 2.6; thus it was expected to find 11.2—but found 20—responses for the correlation between neutral feeling towards technology for the use case diagrams and neutral feeling towards knowledge improvement. Hence more than expected indicated that they have a neutral feeling that technology would improve their knowledge of the subject. Continuing on this statement, the Standardized Residual of 4.2 indicated that it was expected to find 0.4—but found 3—responses in the correlation between 'did not like the use case diagrams' and disagreeing that the technology improved their knowledge. Hence more than expected indicated that they did not like the technology and that it did not improve their knowledge of the subject.

§2 There is a significant association between use case narratives in the interactive textbook and improving the knowledge of the subject. Standardized

Table 2. Cross tabulation w.r.t. the task-technology fit theory

<i>Experience with using the e-textbook to complete assignment for:</i>	<i>Cross tabulation statement</i>	<i>Fisher Exact</i>	<i>Interpretation</i>
Use case diagrams	Improve my knowledge of the subject	<0.001	See §1
Use case narratives	Improve my knowledge of the subject	<0.001	See §2
Creating a holistic view (understanding how systems fits into the real world)	Improve my knowledge of the subject	0.022	See §3
Understanding the scope of the system	Improve my knowledge of the subject	0.001	See §4
Use case diagrams	Improved my contribution to the team	<0.001	See §5
Use case narratives	Improved my contribution to the team	0.006	See §6
Understanding the scope of the system	Improved my contribution to the team	0.006	See §7
The textbook itself	Improved my contribution to the team	0.019	See §8

Residual is 2.4; thus it was expected to find 10.9—but found 19—responses for the correlation between neutral feeling towards technology for the use case narratives and neutral feeling towards knowledge improvement. Hence more than expected indicated that they have a neutral feeling that technology would improve their knowledge of the subject. However the Standardized Residual is -2.0, thus it was expected to find 26—but only found 16—for the correlation between neutral feeling towards technology for the use case narratives and agreeing that their knowledge improved with the technology. Hence less than expected agreed that technology improved their knowledge of the subject. Continuing on this statement, the Standardized Residual of 3.1 indicated that it was expected to find 0.3—but found 2—responses in the correlation between ‘did not like the use case narratives’ and disagreeing that the technology improved their knowledge. Hence more than expected indicated that they did not like the technology and that it did not improved their knowledge of the subject.

§3 There is a significant association between ‘creating a holistic view (understanding how systems fits into the real world)’ and improving the knowledge of the subject.

- §4 There is a significant association between understanding the scope of the system and improving the knowledge of the subject.
- §5 There is a significant association between understanding the use case diagrams and improving the students' contribution towards the team.
- §6 There is a significant association between understanding the use case narratives and improving the students' contribution towards the team.
- §7 There is a significant association between understanding the scope of the system and improving the students' contribution towards the team.
- §8 There is a significant association between the textbook itself and improving the students' contribution towards the team.

From the analysis of above it appears that there are significant associations between the characteristics of the tasks and the technology used to perform the specific task, as well as between the students' understanding of the work, improving his/her knowledge, and contributing to a team. Thus it seems as if our e-textbook did indeed lead to an increased performance by the students—at least as far as their own opinions are concerned. We also saw that generation Z relies strongly on their peers for assistance even though some literature claims that their social skills would be underdeveloped.

5 Conclusion

Generation Z thrives on technology. They are always connected to the world around them, and yet, as educators we often do not realize the potential this connectivity can bring to our courses. If we are needed to guide them in filtering the correct information but also to guide them in challenging them to use their connected time on something that will make them grow and become successful individuals, rather than only purposelessly flipping through various screens and apps.

Bikowski and Casal acknowledged that a large amount of research has already gone into textbook design, but *“little has been done on customised, interactive textbooks designed focusing on specific content”* [3]. With this paper we have responded to the call of [3]. Though our prototype e-textbook is not yet fully-fledged and not yet unanimously accepted by our students, we believe that it is a step in the right direction to connect ourselves and our knowledge with the next generation (Z) of students.

References

1. d'Ambra, J., Wilson, C.S., Akter, S.: Application of the task-technology fit model to structure and evaluate the adoption of E-books by academics. *J. Am. Soc. Inf. Sci. Technol.* **64**(1), 48–64 (2013)
2. Baek, E., Monaghan, J.: Journey to textbook affordability: an investigation of students' use of eTextbooks at multiple campuses. *Int. Rev. Res. Open Dist. Learn.* **14**(3), 1–26 (2013)

3. Bikowski, D., Casal, J.E.: Interactive digital textbooks and engagement: a learning strategies framework. *Lang. Learn. Technol.* **22**(1), 119136 (2018)
4. Bradford, S.: Alternative social media as a recruiting tool for generation Y and generation Z. *Int. J. Innov. Educ. Res.* **6**(10), 253–264 (2018)
5. Chicca, J., Shellenbarger, T.: Connecting with Generation Z: approaches in nursing education. *Teach. Learn. Nurs.* **13**, 180–184 (2018)
6. Chickering, A.W., Gamson, Z.F.: Seven principles for good practice in undergraduate education. *The Wingspread J.* **1–7**, 3–7 (1987)
7. Dennis, A.R., Abaci, S., Morrone, A.S., Plaskoff, J., McNamara, K.O.: Effects of e-Textbook instructor annotations on learner performance. *J. Comput. High. Educ.* **28**, 221–235 (2016)
8. Eckleberry-Hunt, J., Lick, D., Hunt, R.: Is medical education ready for generation Z? *J. Graduate Med. Educ.* **10**(4), 378–381 (2018)
9. Gardner, J.K., Ronzio, C., Snelling, A.: Transformational learning in undergraduate public health education: course design for generation Z. *Pedagogy Health Promot.* **4**(2), 95–100 (2017)
10. Goodhue, D.L., Thompson, R.L.: Task-technology fit and individual performance. *MIS Quart.* **19**(2), 213–236 (1995)
11. Hisrich, R.D., Peters, M.P.: *Entrepreneurship*, 4th edn. McGraw-Hill, New York (1998)
12. Monaco, M., Martin, M.: The millennial student: a new generation of learners. *Athletic Train. Educ. Journ.* **2**, 42–46 (2007)
13. Moore, K., Jones, C., Frazier, R.S.: Engineering education for generation Z. *Am. J. Eng. Educ.* **8**(2), 111–126 (2017)
14. Seemiller, C., Grace, M.: Generation Z: educating and engaging the next generation of students. *About Campus* **22**(3), 21–26 (2017)
15. Shatto, B., Erwin, K.: Moving on from millennials: preparing for generation Z. *J. Contin. Educ. Nurs.* **47**(6), 253–254 (2016)
16. Steyn, R., Millard, S., Jordaan, J.: The use of a learning management system to facilitate student-driven content design: an experiment. In: Huang, T.-C., Lau, R., Huang, Y.-M., Spaniol, M., Yuen, C.-H. (eds.) SETE 2017. LNCS, vol. 10676, pp. 75–94. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-71084-6_10
17. Steyn, R., Botha, A., Mennega, N.: Is a picture truly worth a thousand words? infographics for undergraduate teaching. In: Hao, T., Chen, W., Xie, H., Nadee, W., Lau, R. (eds.) SETE 2018. LNCS, vol. 11284, pp. 69–78. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-03580-8_8
18. Taylor, E., van Aswegen, K.: Students' learning approaches: are they changing? *CCIS* **730**, 37–47 (2017)
19. Turpie, J.: *Creative Engineers*. In: *CreativityMoneyLove: Learning for the 21st Century* (2012)
20. Vikhrova, O.: On some generation Z teaching techniques and methods in higher education. *Information* **20**(9a), 6313–6324 (2017)
21. Ziguers, I., Buckland, B.K.: A theory of task/technology fit and group support systems effectiveness. *MIS Quart.* **22**(3), 313–334 (1998)



Detecting Similarity in Multi-procedure Student Programs Using only Static Code Structure

Karen Bradshaw^(✉)  and Vongai Chindeka

Department of Computer Science, Rhodes University, Grahamstown, South Africa
k.bradshaw@ru.ac.za

Abstract. Plagiarism is prevalent in most undergraduate programming courses, including those where more advanced programming is taught. Typical strategies used to avoid detection include changing variable names and adding empty spaces or comments to the code. Although these changes affect the visual components of the source code, the underlying structure of the code remains the same. This similarity in structure can indicate the presence of plagiarism.

A system has been developed to detect the similarity in the structure of student programs. The detection system works in two phases: The first phase parses the source code and creates a syntax tree, representing the syntactical structure of each of the programs, while the second takes as inputs two program syntax trees and applies various comparison algorithms to detect their similarity. The outcome of the comparison allows the system to report a result from one of four similarity categories: identical structure, isomorphic structure, containing many structural similarities, and containing few structural similarities. Empirical tests on small sample programs show that the prototype implementation is effective in detecting plagiarism in source code, although in some cases manual checking is needed to confirm the presence of plagiarism.

Keywords: Plagiarism detection · Code structure · Student code

1 Introduction

Plagiarism occurs when one person tries to pass off someone else's work as his/her own [6]. This can mean large-scale copy-pasting or merely copying phrases or sentences in the work, which if done without quoting and/or citing the originator of the work, results in plagiarism. This is a common occurrence in academic environments. In undergraduate programming courses, once one student obtains a solution to an assigned programming task, it is often replicated by other students.

Assignments for more advanced programming courses typically involve more complex programs that can be constructed in a variety of ways. Students in these course should be aware of a wider variety of programming constructs available

in the programming language being used. Therefore, if there is similarity in the structure of large segments of code it can be a sign that plagiarism has taken place.

Plagiarism of code often involves techniques that try to hide the plagiarism, such as various code obfuscation techniques. Students commonly resort to simple techniques, such as statement reordering, instruction splitting or aggregation, loop unwinding or introducing white spaces and comments [16]. Although these changes affect the visual appearance of the source code, they do not alter the syntactic structure thereof.

A functional plagiarism detection system is a possible solution to preventing plagiarism in an academic environment by encouraging students to avoid the penalties attached to plagiarism [13]. The main obstacle, however, in detecting plagiarism in an undergraduate programming course is the sheer volume of student programs that need to be assessed. Thus, the probability of detecting similar programs is reduced with larger class sizes and more complex programs.

The aim of this research is to generate a similarity detection system that bases its similarity comparison solely on the static structure of the programs being compared without any pre-processing of the source code or dynamic analysis thereof. Such a system can be useful in the detection of plagiarism in an academic setting and specifically in more advanced programming courses with more complex programming assignments. Such a plagiarism detection system would not be useful in introductory courses, where the structure of the simple programs tends to be much the same even without the occurrence of plagiarism. The effectiveness of the similarity detection system in detecting plagiarism is also investigated.

The rest of this paper is organized as follows: Sect. 2 introduces and discusses related studies in plagiarism detection and tree comparison. Section 3.1 gives a high level overview of the similarity detection system developed. Section 4 explains the various algorithms used in the comparison of the program structure, while Sect. 5 discusses the results of simple test cases. Section 6 concludes the paper and also mentions future work to improve the similarity detector.

2 Related Work

2.1 Plagiarism Detection

Most existing similarity detection systems for source code use either a metrics-driven or syntax-based approach to determine the degree of similarity between programs [6]. The metrics used in the former approach can come from a software engineering perspective, such as the number of each data structure type used or the cyclomatic complexity of the program's control flow. Cyclomatic complexity is a metric based on the number of linearly independent paths in the program's control flow. Metrics can also come from a linguistic or technical aspect such as variable names, indentations and other layout conventions used as well as the number of comments in the code and their quality. These types of metrics can help determine a student's program authoring style. The linguistics centered

metrics tend to be more useful for smaller and simpler programs such as those written in introductory programming courses.

The methods used in systems adopting a syntax-based approach can be categorised into static source code comparison, static executable code comparison, dynamic control flow based, dynamic API based methods as well as dynamic value based methods. JPlag, YAP3 and MOSS are three well-known systems that detect similarities in source code by using a static source code comparison. MOSS (Measure of Software Similarity)¹ is a system developed in 1994 that uses fingerprinting to detect similarity in programs by using a fingerprinting algorithm called winnowing [9]. This algorithm makes detection faster, but at the expense of sacrificing some detection capabilities.

YAP3 [15] works in two phases. In the first phase it removes comments and string constants, changes all letters to lowercase, maps statements that do the same things, reorders the functions in the code to their calling orders by expanding them to their full token sequences and removes all the tokens that are not in the lexicon of the language used. The second phase is the comparison phase, which uses an algorithm that caters for the scrambling of independent segments of code called the Running-Karp-Rabin Greedy-String-Tiling (RKR-GST) algorithm. This is a similar algorithm to that used by the UNIX utility “sdiff”.

In JPlag [7], the first phase scans and parses the program and converts it into token strings based on the program structures. In the second phase the tokens of the two programs are compared using the “Greedy String Tiling” algorithm. Token strings are compared according to the following rules: any token from one program must match with only one token in the other program, substrings are matched without relating to their positions (so that changing the positions of code segments is ineffective) and the matching of long substrings is more indicative than the matching of short substrings so they are favoured more.

The JPlag system is a web-based application so the result is given as a set of HTML pages providing an in-depth description of the similarity. Assuming a pair of programs as input, the system provides results (in the form of a histogram) for each possible match found in the files. Percentages less than 5% do not indicate plagiarism, while a similarity of 100% shows definite plagiarism. Any percentage in between requires further manual investigation to determine if it is plagiarism.

Systems like MOSS, YAP3 and JPlag were designed to work with a number of languages and generally provide good results. However, small changes in the source code such as reordering statements in the case of JPlag and slightly more complex reorderings in the case of MOSS, cannot be detected.

CSPLAG [8] is a solution that attempts to eliminate these shortcomings by using both syntax and semantic knowledge to detect copied code. CSPLAG however, focuses only on languages compiled within the .NET framework. Comparisons of the source code, the abstract syntax trees, as well as the .NET produced intermediate code are carried out to produce results that are superior to those of JPlag and MOSS in detecting a variety of different plagiarism scenarios.

¹ <https://theory.stanford.edu/~aiken/moss/>.

2.2 Tree Comparison Studies

A tree is a special form of a graph, with only one edge connecting any two nodes, that is, without cycles or loops [14]. Tree structures are typically used to represent hierarchical data. A tree is considered rooted if it has a node that is selected to be a root node and is ordered if the children of each node are in a specific order, for example, increasing values of the nodes from left to right.

Isomorphism is a useful concept in comparing trees; two trees are isomorphic if the nodes in one tree can be mapped to the nodes in the other tree [1]. This means that an isomorphic tree can be obtained by switching around the children of the nodes of another tree. For example, the two trees shown in Fig. 1 are isomorphic.

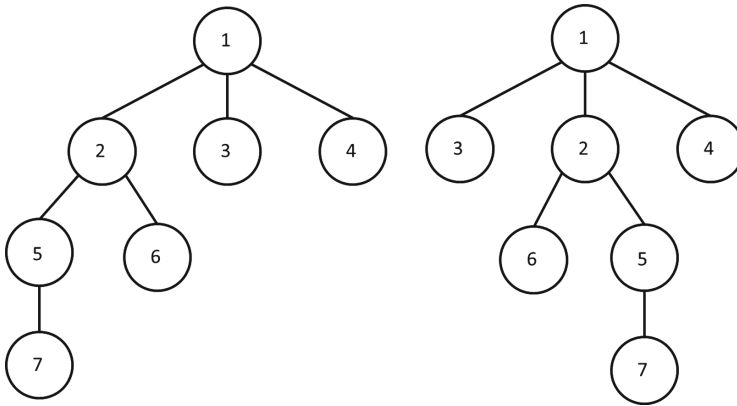


Fig. 1. Example of isomorphic trees

An algorithm to determine whether two trees are isomorphic was developed by Aho et al. [1]. The algorithm applies to trees that are rooted and unordered. In the algorithm, an integer is allocated to each node, beginning at the leaf nodes of the trees, in such a way that the trees are isomorphic if and only if the same integer is assigned to the roots of the trees. This algorithm works in $O(n)$ time for n nodes.

Itokawa et al. [4] proposed an algorithm for tree pattern matching using succinct data structures. Succinct data structures are a representation of data that takes a minimal amount of space but remains usable. This representation (of which there are many forms) is an efficient encoding that does not require decoding so that it may be used in query operations. The succinct representation for trees defined in the algorithm proposed by Itokawa et al. uses matching pairs of parentheses to represent a node's information. This depth-first unary degree sequence (DFUDS) representation is a succinct representation for ordered trees. For a tree with n nodes, the representation is a sequence of $2 * n$ opening and closing parentheses, where an opening parenthesis is emitted when a node is

first encountered, and the closing parenthesis is emitted when returning to this node after the depth first traversal of the respective subtree. Moreover, given the DFUDS representation of two trees p and t , the algorithm returns true if a substitution θ of one tree, called $p\theta$, exists so that t and $p\theta$ are isomorphic.

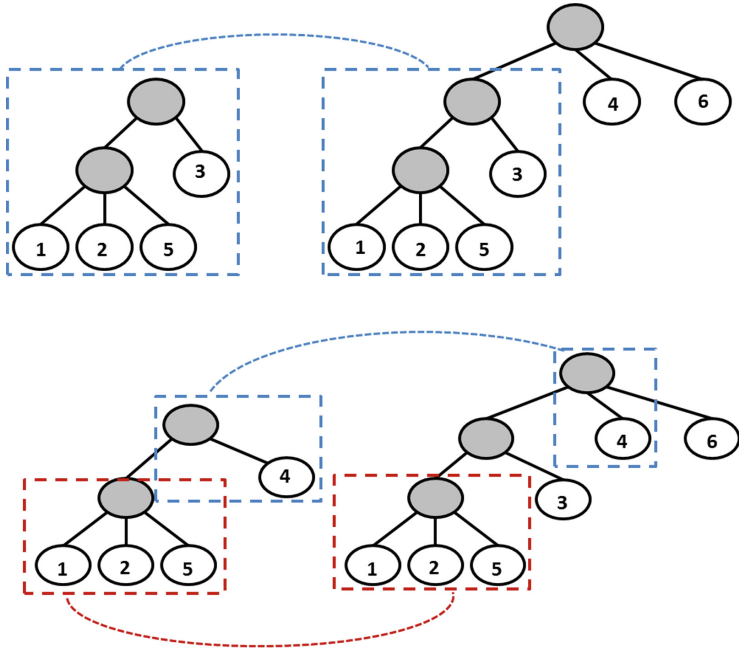


Fig. 2. Graph comparison using maximum agreement subtrees

There are a multitude of graph comparison algorithms based on a variety of methods, including set based, frequent sub-graph based, and kernel based algorithms [10]. Set based algorithms treat graphs as a set of edges and a set of nodes. Candidate graphs are then compared by looking at the similarity of the respective sets. Although these algorithms are readily available, transforming graphs into sets ignores the hierarchical topology thereof. Frequent sub-graph mining algorithms seek to identify sub-graphs that appear frequently in the graphs being compared. Feature selection is then used to isolate the most selective sub-graphs. However, these algorithms have an exponential computational complexity making them less desirable when used with large graphs. Graph kernels change the topology of a graph in a way that does not ignore what it represents. They then compare substructures of the graphs in polynomial time. Kernels have been developed that look at different substructures such as subtrees and shortest paths.

This research makes use of one of the so-called consensus comparison methods that computes a new tree containing the maximum agreement subtree (MAST)

of the trees being compared. The MAST is a tree that includes all possible matching nodes of the trees based on common ancestors, as illustrated in Fig. 2. Since the problem was first posed by Finden and Gordon in 1985 [3], much research has been focused on solving the MAST problem. One of the best known and fastest algorithms for finding a MAST was proposed by Bryant [2] with a reported time complexity of $O(kn^3 + n^d)$. Kao et al. [5] presented an algorithm for tree comparison using the MAST, which was claimed to be faster than the existing ones. However, their algorithm was applicable to graphs with labelled nodes only, although the method for labelling the graphs was not restricted to any specific way for the algorithm to work. More recently, Wang and Swenson [12] extended Bryant's algorithm to reduce both the time complexity and the number of MASTs found by introducing the kernel agreement subtree, computed as the intersection of the MASTs.

3 Proposed Similarity Detection System

3.1 Design Overview

For the prototype implementation of the proposed detection system, an experimental language Parva [11] with a small set of programming constructs, was used as the source language. Similar to YAP and JPlag, the proposed system comprises two phases, but unlike these systems, syntax trees are used as the representation of the source programs.

In the first phase, a parser translates the given input programs into their respective syntax trees, which are subsequently input to the second phase, where various comparison algorithms are applied to determine the similarity score of the input programs.

To implement the first phase, a Parva parser, written in C#, was developed using a compiler generator, Coco/R². The output from this parser is a concrete syntax tree, which is serialised and stored in eXtensible Markup Language (XML) format. XML was chosen due to its suitability for use in phase two of the similarity detection system, as well as for ease of viewing the tree structures during manual validation of the similarity of the test programs. Although there are C# libraries that implement XML serialising and deserialising, the proposed system uses a custom implementation, which allowed a more flexible and accurate representation of the hierarchical format of the tree structures.

The similarity detection phase takes in the XML representations of the Parva source code files and deserialises these into the `Tree<string>` data structure form consisting of a collection of nodes of type `TreeNode`. Each node has a value field of a generic type, a property indicating the level in the tree where the node is located, a pointer to its parent node and finally, a list of its children nodes.

During the deserialisation, a new node is created by providing the value of the node as well as the node's parent node. This creates a level 0 node with the given value and parent, as well as an empty list of children. Functionality for

² <http://www.ssw.uni-linz.ac.at/coco/>.

adding a child to the node, using the `AddChild(TreeNode<T> child)` method, is provided for. The tree can also be represented in a string form using the `ToString()` method, which does a depth first traversal of the tree rooted at the current node and returns a string containing the values of the node itself, its parent node and its children nodes, for each node that is encountered during the traversal. This string form is a version of succinct representation that is useful because simple string handing operations can replace handling bulky tree data structures during the comparison.

The resulting trees are compared using three algorithms: a brute force algorithm, an isomorphism algorithm as well as an algorithm based on succinct representation. An overview of the similarity detection phase is illustrated in Fig. 3.

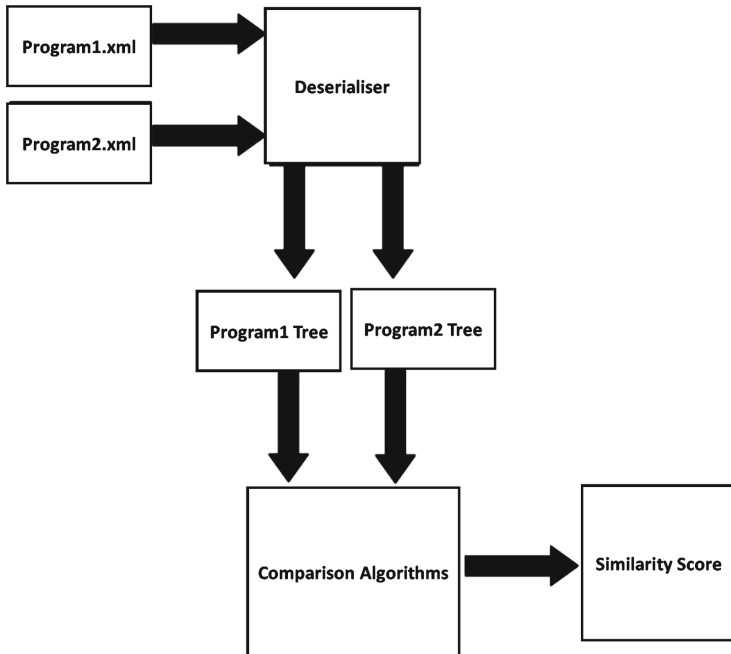


Fig. 3. Similarity detection phase

3.2 Example Program

The minimal example Parva code in Listing 1 contains a main function with only one component, a constant variable declaration. The string representation of the parse tree generated by parsing this declaration, shown in Listing 2, gives a list of the nodes of the tree iterated through in a depth-first order. Each node is represented by the level and value of the node followed by the value of the parent node as well as values of the children nodes if they exist. The XML representation of the parse tree is illustrated in Fig. 4.

```
void Main ()
{ const votingAge = 18; }
```

Listing 1. Minimal Parva program

```
0 Node Value: Program Parent Value: null Children:
  FuncOrGlobalVarDeclarations
1 Node Value: FuncOrGlobalVarDeclarations Parent Value: Program
  Children: Type Identifier Function
2 Node Value: Type Parent Value: FuncOrGlobalVarDeclarations
  Children: Void
3 Node Value: Void Parent Value: Type Children:
2 Node Value: Identifier Parent Value:
  FuncOrGlobalVarDeclarations Children:
2 Node Value: Function Parent Value: FuncOrGlobalVarDeclarations
  Children: FormalParameters Body
3 Node Value: FormalParameters Parent Value: Function Children:
3 Node Value: Body Parent Value: Function Children: Statement
  Statement Statement Statement Statement Statement
4 Node Value: Statement Parent Value: Body Children:
  ConstDeclarations
5 Node Value: ConstDeclarations Parent Value: Statement Children:
  OneConst
6 Node Value: OneConst Parent Value: ConstDeclarations Children:
  Identifier AssignOp Constant
7 Node Value: Identifier Parent Value: OneConst Children:
7 Node Value: AssignOp Parent Value: OneConst Children:
7 Node Value: Constant Parent Value: OneConst Children:
  IntegerConstant
8 Node Value: IntegerConstant Parent Value: Constant Children:
```

Listing 2. String representation of the syntax tree for the minimal Parva program

4 Comparison Algorithms

The report on the similarity output by the similarity detection phase involves one of four categories: category 1 (identical), category 2 (isomorphic), category 3 (contains many similarities) and category 4 (contains few similarities). The three chosen comparison algorithms give results that classify the program similarity into these categories.

In addition to the basic string representation, a succinct representation was also created for use in some of the comparison algorithms. As discussed in Sect. 2.2, a succinct data structure represents the data in a way that takes up minimal space, but allowing operations on the data to be possible without the requirement of decoding the data. The succinct representation that was chosen is a string that represents each node as a pair of opening and closing parentheses preceded by the value of the node and the succinct representations of the children nodes, enclosed in the parentheses. A depth-first traversal is done through

```

- <Program>
  - <FuncOrGlobalVarDeclarations>
    - <Type>
      <Void> </Void>
    </Type>
    <Identifier> </Identifier>
  - <Function>
    <FormalParameters> </FormalParameters>
  - <Body>
    - <Statement>
      - <ConstDeclarations>
        - <OneConst>
          <Identifier> </Identifier>
          <AssignOp> </AssignOp>
          - <Constant>
            <IntegerConstant> </IntegerConstant>
          </Constant>
        </OneConst>
      </ConstDeclarations>
    </Statement>
  </Body>
</Function>
</FuncOrGlobalVarDeclarations>
</Program>
    
```

Fig. 4. XML representation of part of the parse tree for the minimal Parva program

the tree, starting at the root. When a leaf node is reached, its representation is passed back up the tree resulting in the encoded version of the tree. For the example tree shown in Fig. 5, the resulting encoding is: 1(2(4(5())6())7())8()).

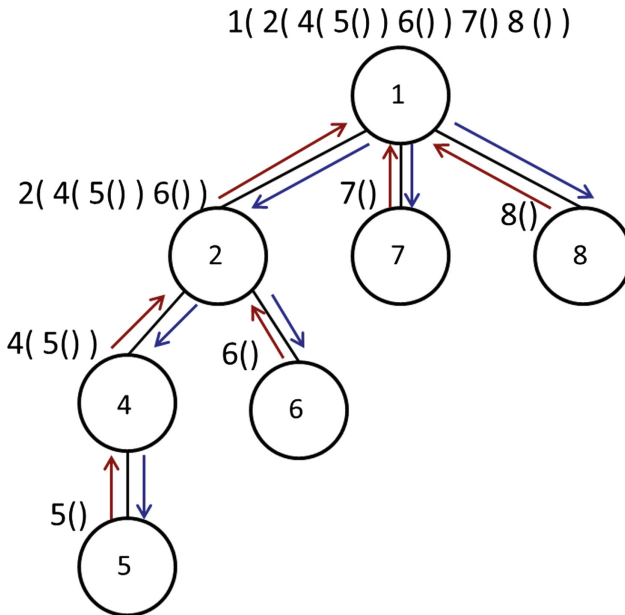


Fig. 5. Example showing use of the succinct representation

The succinct string encoder recursively does a depth-first traversal of the tree returning the encoding of each node. The result is a string, which means that normal string operations can be applied to the representation. This representation takes up less space, compared with the basic string and XML representations of the syntax tree.

4.1 Brute Force Comparison

The brute force algorithm compares the basic string representations of the trees rooted at the given nodes; it returns true if the strings are identical and false if they are not. The string representation is obtained using a method that recursively does a depth-first traversal of the tree rooted at the current node. The depth-first traversal means that if the resulting strings are identical, the trees that are traversed are identical. A `true` result output by this algorithm gives a category 1 result.

4.2 Isomorphism

The succinct string encoder first sorts the children of a node before recursively traversing them. This is done so that the rearranging of statements, which results in the rearranging of nodes in the tree, is ineffective in obscuring the similarity. This sorting is done using a method that compares two nodes by checking if the basic string representing the current node is alphabetically less than, equal to or greater than the string representation of the next node. This means that the children of a node are sorted according to the alphabetical order of the respective string representations.

What is achieved by sorting the children before traversing through them, is that the order of the nodes in the tree is inconsequential. Trees are isomorphic if one tree can be obtained by switching the order of siblings, that is, nodes at the same level. By removing order as a factor, the isomorphism of the trees can be exposed. Given the root nodes of two trees being checked for isomorphic similarity, the algorithm encodes both the trees and checks if the resulting trees are identical. Classifying the trees as being isomorphic means that the two trees contain the same elements structurally and if they do not return true for the brute force algorithm it means that the statements of one program were rearranged to produce the other program.

4.3 Individual Node Comparison

If the programs being compared have not been declared identical or isomorphic, additional comparisons using the succinct representation are carried out. Consider two syntax trees `temp0` and `temp1`, where `temp0` has fewer nodes than `temp1`. First `temp1` is encoded and for each node of `temp0`, the encoding of `temp1` is checked to see if it contains an encoding of the node. If the encoding of one tree contains the encoding of a node in the other tree it means that a similar

node has been found in the trees. The system keeps track of all the similar nodes and if the node is a global level node, that is, either a global variable declaration or function declaration, this is also noted.

Based on the percentage of similar nodes that are found, a category 3 or category 4 result is given. A category 3 result is given for similarity greater than 60% (more than 60% of nodes are similar) for either tree. A category 4 result is given for similarity less than 60%. For either category, if a global level node is similar it is reported as such. The threshold percentage between categories 3 and 4 was decided arbitrarily, purely to distinguish the two categories.

5 Experimental Results

The measure of similarity that is used to give the result, is based on four categories as mentioned in Sect. 4. Listing 3 shows an example of the result produced by programs that are structurally the same and return `true` for the brute force algorithm described in Sect. 4.1. The Parva translator disregards white spaces and comments. When the syntax tree is generated by the parser, identifier names, string literals, character literals and other values are disregarded. This means that making simple visual changes, such as changing identifier names and string literals as well as adding or removing comments or white spaces, does not trick the similarity detection system.

```
Result
Category 1:
Programs are identical
Similarity is 100%
```

Listing 3. Output showing a category 1 result

Listing 4 shows an example of the result produced by trees that are isomorphic. This means that the obfuscation technique of rearranging statements does not prevent the similarity check from working correctly.

```
Result
Category 2:
Programs are the same; statements have been switched around
Similarity is 100%
```

Listing 4. Output showing a category 2 result

```
Result
Category 3:
The programs contain large/many similar parts
Program 1 (the 1st argument) contains: 85.06% similar nodes
Program 2 (the 2nd argument) contains: 52.85% similar nodes
```

Listing 5. Output showing a category 3 result

Additional comparisons using the succinct representation results in either a category 3 result shown in Listing 5 or a category 4 result shown in Listing 6. Category 3 and 4 results also record the existence of a global level node that is structurally the same, if present.

```
Result
Category 4:
The programs contain few similar parts
At least one global level element (function of variable) is the
same
Program 1 (the 1st argument) contains: 32.51% similar nodes
Program 2 (the 2nd argument) contains: 53.70% similar nodes
```

Listing 6. Output showing a category 4 result

```
void voter()
{ int votingAge = 18;
  writeLine("Voting age = ", votingAge);
}

void voter1(){
// voter.pav
// Simple voter example
voter(); //Write voting age
const votingAge = 18;
int age, eligible = 0, total = 0;
bool allEligible = true;
int[] voters = new int[100];
read(age);
while (age > 0) {
  bool canVote = age > votingAge;
  allEligible = allEligible && canVote;

  if (canVote) {
    voters[eligible] = age;
    eligible = eligible + 1;
    total = total + voters[eligible - 1];
  }
  read(age);
}
if (allEligible) write("Everyone was above voting age");
write(eligible, " voters. Average age is ", total /
  eligible, "\n");
}

void Main ()
{ voter1(); }
```

Listing 7. Test0.pav source code

5.1 Test Cases

Experiments using three source code files were used to validate the results given by the system. The first file used is `Test0.pav` shown in Listing 7. The second file, `Test1.pav` shown in Listing 8, was produced by copying the code in the first program and applying obfuscation techniques that rely on semantic meaning such as, changing the type of loop used, expanding variable declarations and assignments and using a chain of redundant functions to call a function.

```

void voter()
  { writeLine("Voting age = ", 18); }

void voter1(){
// voter.pav -- Simple voter example
  voter(); //Write voting age
  int votingAge = 18;
  int age, eligible = 0, total = 0;
  bool allEligible;
  allEligible = true;
  int[] voters;
  voters = new int[100];
  read(age);
  loop {
    bool canVote = age > votingAge;
    allEligible = allEligible && canVote;
    if (canVote) {
      voters[eligible] = age;
      eligible = eligible + 1;
      total = total + voters[eligible - 1];
    }
    read(age);
    if(age < 0) break;
  }
  if (allEligible) write("Everyone was above voting age");
  write(eligible, " voters. Average age is ", total /
    eligible, "\n");
}

void calling() { voter1(); }

void calling1() { // spurious code
  calling(); }

void calling2() { // more useless code
  calling1(); }

void Main()
  { calling(); }

```

Listing 8. Test1.pav source code

The third file, `Test2.pav` shown in Listing 9, was obtained by applying obfuscation techniques that affect the visual appearance of the code, such as changing white spaces and comments, changing variable names and shuffling statements around or hiding blocks of copied code in other functions.

```

void validVoters() {
// validVoters -- Simple voter example
  const AgeOfVoting = 18;
  int age, eligibleCount = 0, total = 0;
  bool allEligible = true;
  int[] voters = new int[100];
  writeLine("Voting age = ", AgeOfVoting); //Output voting age
  read(age);
  while (age > 0) { //Descriptions
    bool canVote = age > AgeOfVoting;
    allEligible = allEligible && canVote;
    if (canVote) { //Descriptions
      voters[eligibleCount] = age;
      // Arbitrary comments
      eligibleCount = eligibleCount + 1;
      total = total + voters[eligibleCount -1];
    }
    read(age);
  }
  if (allEligible) write("Everyone was above voting age");
  write(eligibleCount, " voters. Average age is ", total /
    eligibleCount, "\n");
}

void Main ()
{ validVoters(); }

```

Listing 9. Test2.pav source code

```

Result
Category 3:
The programs contain large/many similar parts
At least one global level element (function of variable) is the
same
Program 1 (the 1st argument) contains: 90.05% similar nodes
Program 2 (the 2nd argument) contains: 82.27% similar nodes

```

Listing 10. Result of comparing Test0.pav and Test1.pav

The output given when comparing `Test0.pav` and `Test1.pav` is shown in Listing 10. `Test0` has 90.05% of its nodes similar to those in `Test1`, while 82.27% of `Test1`'s nodes are similar to those in `Test0`.

Listing 11 shows the results when comparing `Test0.pav` and `Test2.pav`. `Test0` has 89.05% similar nodes while `Test2` has 96.24% of its nodes the same. The average of the percentages given indicates that the similarity is higher in the `Test0` and `Test2` comparison than the `Test0` and `Test1` one.

```

Result
Category 3:
The programs contain large/many similar parts
At least one global level element (function of variable) is the
same
Program 1 (the 1st argument) contains: 89.05% similar nodes
Program 2 (the 2nd argument) contains: 96.24% similar nodes

```

Listing 11. Result of comparing Test0.pav and Test2.pav

5.2 Discussion of the Results

As is the case for JPlag [7], a result of 100% similarity such as that given in categories 1 and 2 is indicative of plagiarism whereas low percentages of similarity show the absence of plagiarism. For results in between, however, a manual check is required to confirm the presence of plagiarism. Category 3 and 4 results, therefore, require manual intervention.

The test cases shown above compare the effects of semantic-based and visual-based obfuscation techniques on the system. Semantic-based obfuscation techniques appear to be detected correctly by the system; however, in these examples, the presence of similar nodes may also be due to the minimal number of constructs available in the Parva programming language, or to the simplicity of the examples used. The same statements may be chosen by programmers due to lack of choice of alternative constructs, resulting in unintentional similarity. The percentage of similarity detected is generally higher for the visual-based techniques; this is most likely because semantic-based techniques change the structure of the program and require the inclusion of semantic meaning in the process of detecting similarity.

6 Conclusion

The aim of this research was to produce a prototype similarity detection system that compares programs for similarity using only the syntactic structure of the programs. The system consists of a parsing phase, which outputs a syntax tree represented as XML, and a comparison phase. The XML tree representation is converted into both a string and a succinct representation for use in the second phase.

The usefulness of this prototype system in detecting plagiarism was tested using simple, yet realistic student code examples. Results for all four categories of plagiarism were correctly given. Thus, the system could be used both to detect 100% similarity as well as to narrow down the submissions from a large group of students that need to be manually checked.

Future work involves extensive testing on larger code samples and samples taken from real programming languages, as well as optimising the comparison, serialising and encoding algorithms to make the system more efficient. In addition, comparisons with other plagiarism detection systems need to be completed.

References

1. Aho, A.V., Hopcroft, J.E., Ullman, J.D.: The Design and Analysis of Computer Algorithms. Addison-Wesley, Boston (1974)
2. Bryant, D.: Building trees, hunting for trees, and comparing trees: theory and methods in phylogenetic analysis. Ph.D. thesis, University of Canterbury (1997)
3. Finden, R., Gordon, A.: Obtaining common pruned trees. *J. Classification* **2**, 255–276 (1985)
4. Itokawa, Y., Wada, M., Ishii, T., Uchida, T.: Tree pattern matching algorithm using a succinct data structure. *Proc. Int. MultiConf. Eng. Comput. Sci.* **1**, 206–211 (2011)
5. Kao, M.Y., Lam, T.W., Sung, W.K., Ting, H.F.: An even faster and more unifying algorithm for comparing trees via unbalanced bipartite matchings. *J. Algorithms* **40**(2), 212–233 (2001). <https://doi.org/10.1006/jagm.2001.1163>
6. Paris, M.: Source code and text plagiarism detection strategies. In: 4th Annual Conference of the LTSN Centre for Information and Computer Sciences, pp. 74–78. LTSN Centre for Information and Computer Sciences (2003)
7. Prechelt, L., Malpohl, G., Phillippsen, M.: JPlag: finding plagiarisms among a set of programs. Technical report, Karlsruhe Institute of Technology (2000)
8. Puflović, D., Gligorijević, M.F., Stoimenov, L.: CSPlag: a source code plagiarism detection using syntax trees and intermediate language. In: Proceedings of the 52nd International Scientific Conference on Information, Communication and Energy Systems and Technologies (ICEST 2017), pp. 102–105 (2017)
9. Schleimer, S., Wilkerson, D.S., Aiken, A.: Winnowing: local algorithms for document fingerprinting. In: Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data, pp. 76–85. SIGMOD 2003, ACM, New York, NY, USA (2003). <https://doi.org/10.1145/872757.872770>
10. Shervashidze, N., Vishwanathan, S., Petri, T., Mehlhorn, K., Borgwardt, K.: Efficient graphlet kernels for large graph comparison. In: Artificial Intelligence and Statistics, pp. 488–495 (2009)
11. Terry, P.: Compiling with C# and Java. Pearson Education, London (2005)
12. Wang, B., Swenson, K.M.: A faster algorithm for computing the kernel of maximum agreement subtrees. *IEEE/ACM Trans. Comput. Biol. Bioinform.* **1** (2019). <https://doi.org/10.1109/TCBB.2019.2922955>
13. Whale, G.: Identification of program similarity in large populations. *Comput. J.* **33**(2), 140–146 (1990). <https://doi.org/10.1093/comjnl/33.2.140>
14. Wilson, R.J., Watkins, J.J.: Graphs: an Introductory Approach: A First Course in Discrete Mathematics. John Wiley & Sons Inc, Hoboken (1990)
15. Wise, M.J.: YAP3: improved detection of similarities in computer program and other texts. *SIGCSE Bull.* **28**(1), 130–134 (1996). <https://doi.org/10.1145/236462.236525>
16. Zhang, F., Wu, D., Liu, P., Zhu, S.: Program logic based software plagiarism detection. In: IEEE 25th International Symposium on Software Reliability Engineering (ISSRE), pp. 66–77. IEEE (2014)



Enhancing Computer Students' Academic Performance Through Explanatory Modeling

Leah Mutanu  and Philip Machoka  

Department of Information Systems,
United States International University Africa, Nairobi, Kenya
{[lmutanu](mailto:lmutanu@usiu.ac.ke), [pmachoka](mailto:pmachoka@usiu.ac.ke)}@usiu.ac.ke

Abstract. A key challenge facing nowadays universities is the growing attrition rate of computer studies students, attributed to poor academic performance. While extensive research has been conducted on how to enhance students' performance in computer programming, fewer research investigates other computer courses, especially in sub-Saharan Africa. This paper addresses this gap by describing experiments that revealed some of the factors that influence a student's overall academic performance at university through explanatory modeling. Our results showed that students' background in mathematics and their performance in the Introduction to Information Systems course were key in determining performance. Unexpectedly, prior computer skills or secondary school grades had less impact. The strategies identified for enhancing students' performance include an emphasis on building students' mathematics background, providing a stringent teaching approach to foundational computing courses, re-structuring of courses in the computer program, and linking courses across the curriculum. Thus, explanatory modeling creates an opportunity to adopt a proactive approach to enhancing the performance of computer studies students.

Keywords: Computer science higher education · Academic performance · Explanatory modeling

1 Introduction

Globally, Higher Education is the fastest growing segment of post-secondary education. However, this sector faces a myriad of challenges. Key among them is *student retention*. The challenges, however, vary from region to region. In the USA, for example, the challenge is retaining more women and people from under-represented minorities (Afr.-Am., Hisp., Native Am.) in computer-related studies. In Kenya, by contrast, *education quality* is cited as one of the key challenges facing public university education [19]. Universities also face low enrolment numbers in computer-related degree programs as compared to other degree programs. Another challenge is the high number of student dropout rates.

While various reasons exist why students drop out, students' academic performance has been identified as one of the biggest drivers [8]. A study by Njoroge (et al.) on student attrition rates in private universities in Kenya showed that academic performance contributed to increased attrition rates [18]. The study recommended mechanisms to be put in place for early detection of *attrition risk* supported by technology to ensure students pursue their studies to completion; for comparison see [3].

Our objective was to pursue [18]'s recommendation, that is, to identify factors in students' learning environment that can serve as indicators of students' academic performance. We focused on academic factors that impede students' performance in computer-related studies. The remainder of the paper is organized as follows: Sect. 2 outlines related work; Sect. 3 discusses our research method; Sect. 4 presents the findings and discussions; Sect. 5 highlights the main limitations of our study; Sect. 6 concludes and makes recommendations for future work.

2 Related Work

High dropout rates are common in Computer-related degree programs at universities. Two main causes of the problem pointed out by [7] are students' motivation and the complexity of these courses. In determining these problems, however, [7] did not consider the students' prior education background before university. The assumption was that any student who has enrolled in a computer-related program was qualified to undertake the course. However, our study shows that there are prior academic background factors that can influence a student's academic performance. This study did not consider students' social, economic, cultural, and geographic factors, because their treatment requires a different approach. Kumar found that different demographic groups (economic status, gender, race, major, and type of institution) required different intervention approaches in Computer Science [13]. This study, therefore, attempts to explore students' prior academic background before university rather than their demographic groups.

Al Murtadha (et al.) investigated the key factors that influenced ICT students' academic performance in Saudi Arabia [2]. A number of factors were identified, including age, gender, student major, means of transport to school, parents' education level, English proficiency level, sitting position, fear of exam, study schedule, drug abuse, daily sleeping hours, Twitter use, sports engagement, hobbies, and community service engagements. These factors can be categorized into 'social', 'economic', and 'academic'. While all those factors are significant, our study focuses only on those factors that academic institutions themselves can influence, namely the academic factors, through the provision of remedial courses or alternative teaching approaches. These factors include the students' academic background in Mathematics, Computer and English courses. Nash pointed out that students joining tertiary education are not always sufficiently equipped [17]. For example, university students are expected to use computers to access course materials, write assignments with good grammar, and to carry out calculations.

To be able to cope with these demands, students need a general competence in computer usage, language, and mathematics. Additionally considered in our study is the students' academic performance at university.

A study by Garcia and Al-Safadi on factors affecting students' academic performance in computer programming found that classroom management skills were key to improving student performance [6]. Their study, however, focused only on the performance in computer programming—not the entire curriculum. More recently, Wang (et al.) conducted a similar study that also focused on enhancing students' computer programming performance [24]. They recommended a more comprehensive investigation focused on other courses in the degree program, rather than only on one subject. While there are many publications on how technology enhances students' performance in general, little research focuses on how to enhance students' performance in computer-related studies (ICT). To the best of our knowledge, most of the existing papers in this area identified factors influencing only a single subject, computer programming [11], ignoring other subjects in the curriculum. Our work addresses this gap by looking at performance in all courses in our entire degree program. Furthermore, there are limited studies in the area focusing on institutions in sub-Saharan Africa.

The results of [6] were based on students' own perceptions. Accordingly, instructors' classroom management skills, such as preparation for the topic and teaching techniques, influenced performance. Instructional materials did not have a big influence. A related study that also sought students' opinions indicated that education pedagogy (didactics) was a key factor in influencing students' performance in computer-related courses [20]. In [4], Barlow-Jones and Westhuizen show the relationship between university pre-entry attributes and students' performance in computer programming: There was a correlation between previous programming experience and performance in programming modules while there was no correlation between the socio-economic status, educational background, highschool Mathematics, and English scores. Their study, however, also collected data from students' opinions rather than examining actual students' scores. Our research follows an approach that does not focus on students' perceptions but on their actual performance. Thus we hope to paint a more accurate picture of the background factors that influence performance.

Computer students often require a special set of digital skills that other degree programs do not demand. This is especially challenging in developing countries where the 'digital divide' gap is large. The lack of infrastructure and low household income often deny students the opportunity to engage with technology adequately especially during their formative years of schooling. Chikumba highlights the extent of the problem in Malawi where private secondary schools performed better in computer studies than public secondary schools due to poor investment in computers, teaching materials, and staff required to deliver the subject [5]. For this reason it is not a requirement for students joining many universities in developing countries offering computer-related degree programs to have this prerequisite technical background. However, some courses can be

recommended for students intending to take computer related studies. For example, Akinola and Nosiru recommend *a priori* knowledge of Physics and Mathematics as essential for students to excel in computer-related studies [1]. They also mention that better teaching methods and techniques can enhance students' performance by changing their perception of computer-related courses. A different study on factors promoting success in Computer Science revealed the Computer Science performance predictive factors as the students' comfort level and mathematical background [25]. No significant difference was based on gender. The performance in the introduction to programming courses also had a positive influence on success. Research on how language influences performance revealed that—contrary to the generally accepted view that achievement in highschool mathematics courses is the best individual predictor of success in undergraduate Computer Science—success in English correlates better with the actual performance [22]¹. Importantly, their study was conducted in a social context in which many students were not native English speakers.

In [1], Akinola and Nosiru used a *fuzzy sets* approach to yield the factors that influence performance. They posit that selecting the factors that influence students' performance in computer programming is not an easy task as it involves human decision-making which can be imprecise or subjective. This problem cannot be handled effectively by probability theories. Fuzzy sets are suitable for problems that involve the need to seek consensus among many decision makers. Students gave their opinions according to their own criteria for each factor by selecting a value. The union of their evaluations to all the currently available alternatives was represented in the form of a fuzzy set. They state that the approach helped to eliminate outlier decision-making which lead to a more accurate and reliable result. Although faculty members (instructors) have traditionally found ways of identifying performance challenges 'intuitively', there is a need to enhance this process through innovative ways of modeling data. The objective of the research was to model student data, captured over a period of time, with the aim of providing causal explanations of (poor) performance for purposes of early intervention. Data mining techniques have been used extensively to develop early warning systems for risk aversion. In the health domain, for example, early warning systems have been defined as surveillance systems that collect information on epidemic-prone issues such as diseases in order to trigger prompt public interventions [16]. Fuzzy logic is used to map risk patterns. Our work therefore aimed at modeling computer students' academic performance in order to find ways of enhancing it.

3 Method

Our study was conducted at a private university in Kenya. This university offers two computer-related degree programs, *Computer Science* and *Information Systems*. Both had the same pre-entry admission requirements for secondary school

¹ I.e.: being able to read and understand textbooks and assignment specifications written in high-level academic language.

final exam grade point average (GPA); however the Computer Science program required higher secondary school scores in Mathematics and Physics. Secondary school data was collected on students taking these programs for a period of five years. Only data recorded in the university's Student Information System (SIS) could be used—i.e.: the students' GPA at secondary school, placement test scores (skills assessment on admission to university), chosen major subject, gender, current year of study, year of admission, current GPA for courses taken so far at the university, and the country of origin. We focused on academic factors that the institution itself could influence or use to influence practices at the university to enhance the students' academic performance. We aimed at modeling computer students' academic performance in order to find ways of enhancing it. We achieved this through two *research objectives*:

1. Identification of academic factors influencing students' academic performance in computer-related studies;
2. Establishing ways of enhancing the computer-related degree curricula to address academic factors influencing students' academic performance in computer-related studies.

Our data set related to 6000 students who had joined the university over a five-year period (2014–2018). A random sample was required for analysis. We used the following formula to calculate the study's sample size S as:

$$S = \frac{\frac{z^2 xp(1-p)}{e^2}}{1 + \frac{z^2 xp(1-p)}{e^2 N}}$$

whereby N = population size, e = margin of error (percentage in decimal form), z = the statistical Z-score,² and p = sample proportion. A confidence level of 95% was aimed for, which requires a Z-score of 1.96. A 3% margin of error was chosen. The sample size was therefore 907. Therefore we picked 1000 entries from a population of 6000 records. This sample was picked as a test set for experiments in predictive modeling. However, future experiments shall use the entire sample size. Through the data cleaning process, several records were discarded as not suitable, yielding a final sample size of 858 entries. This number was still acceptable, as it still yielded the desired 3% margin of error.

For ethical purposes our data set was anonymized. An SQL script was written to extract only the necessary data from the data base. The resultant data was saved in an excel spreadsheet for 'cleaning' and subsequent analysis. To ensure that the data was an accurate representation of the five years, the year of admission was also retrieved. During 'cleaning' we eliminated 142 records that contained obvious errors or had missing (empty) fields. Data analytics was then conducted with the data mining tools Weka [9] and R [21]. The IBM SPSS statistical analysis tool [14] was also used to explore and validate the results. The use of several tools enriched our insights in validating the results obtained. Clustering and decision tree algorithms were used to classify the data. From the data

² <https://www.investopedia.com/terms/z/zscore.asp>.

mining exercise, patterns in the data were identified and used for explanatory modeling. Our research design was descriptive where the characteristics of correlations between two or more entities were explored and visualization techniques were used to represent the data. Quantitative research techniques were applied to emphasize objective measurements. In our work, data was gathered and used to generalize across groups of students to explain academic performance.

4 Findings

This section identifies patterns in the data for purposes of modeling the performance of students taking undergraduate computer-related degree programs at our university. We describe the techniques used and the results obtained³.

4.1 Factors Influencing Cumulative GPA

Our research set out to identify some academic factors that influence students' performance based on the literature which we have reviewed. Specifically we analyzed factors such as the students' final GPA at secondary school, placement test scores on admission to university, students' chosen major, gender, current year of study, year of admission, test scores for courses done at university, and the country of origin. Placement tests were given to students upon joining the university. Scores from these tests are used to assess students' prior knowledge in Mathematics, English, and computing skills upon admission. Students who do not meet the requirements must take remedial classes for a semester in the respective courses. The prior mathematics knowledge that computer programs require include basic algebra and statistics. The degree programs also offer additional mathematics courses such as discrete mathematics and algebra. Most of the computer courses offered at the university have Mathematics courses as prerequisites. These include courses such as data structures, decision analysis, and data analytics. However, the first computer programming course, Fundamentals to Programming Logic, assumes no prior knowledge of programming and does not have any mathematics prerequisites. The focus of the course is on imparting procedural programming skills.

Distribution of the frequencies for the data used in this study is shown in Fig. 1. The figure shows that there were more male students in both programs. Given a 'pass' mark of 60%, the majority of the students were able to pass the placement tests in IST (computer placement) and English. The performance in Mathematics was not as good, however, it should be noted that *Computer Science competes with other Science and Engineering disciplines for students with a good Mathematics Background.*

³ Due to shortage of page-space in this conference paper we cannot reproduce in this section all the data tables and graphical figures which our research has yielded. Readers who wish to obtain those tables and figures, which are not shown in this paper, may contact us via e-mail.

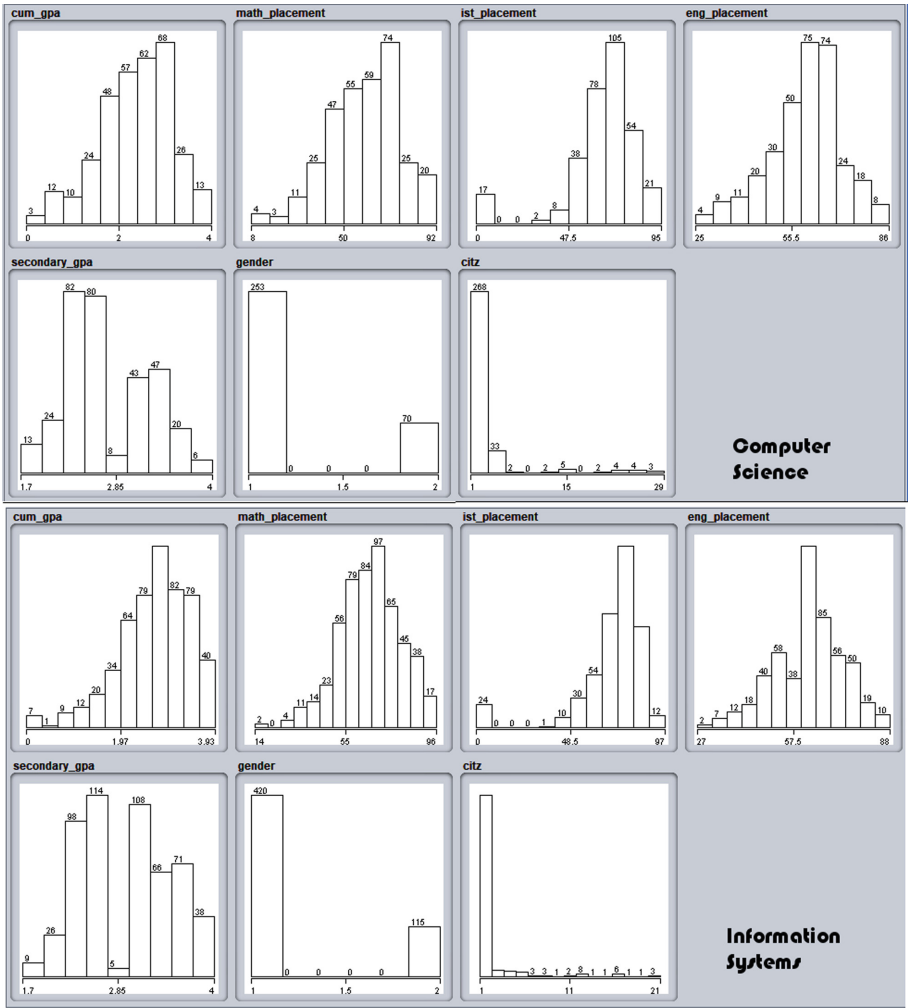


Fig. 1. Sample frequency distributions of the data sets used

The data was analyzed using clustering algorithms to give an indication of which factors influenced the cumulative GPA. For this paper the Expectation Maximization (EM) clustering technique of [10] was preferred, because it provides better optimization than distance-based or hard membership algorithms like K-Means [15]. EM easily accommodates categorical and continuous data fields, thus making it the most effective technique available for proper probabilistic clustering. K-Means clustering is a method of vector quantization that partitions observations into clusters based on the nearest mean, serving as a prototype of the cluster. K-Means was also used for purposes of validating the results. The data had to be converted into numeric representations to perform K-mean clustering.

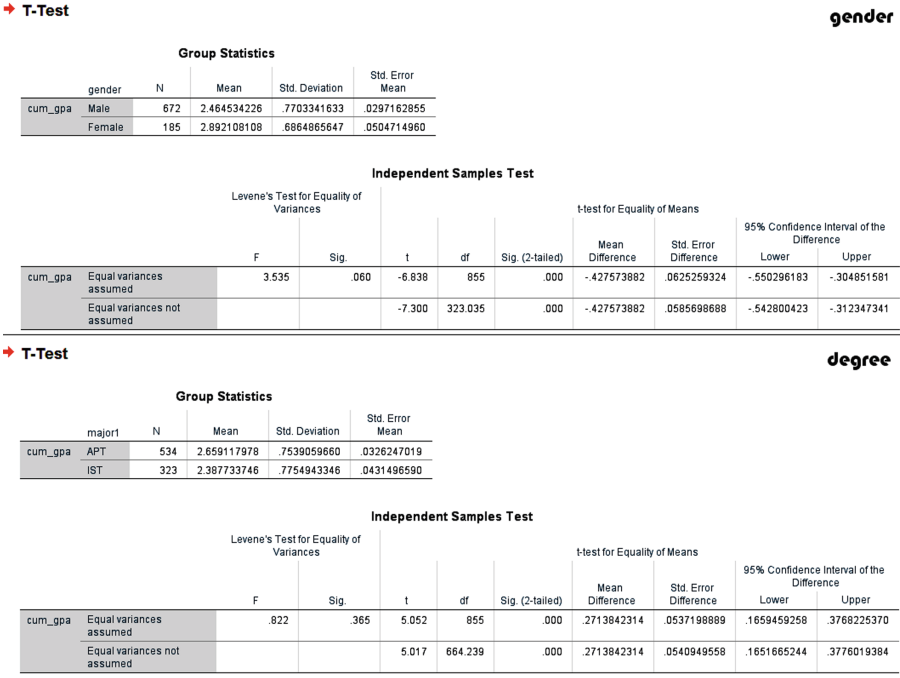


Fig. 2. Significance analysis for gender and degree program

Cluster analysis showed that some variables have a bigger impact on the students' cumulative GPA than others. For example, a higher math placement score resulted in a higher students' cumulative GPA. Both EM and K-Means clustering algorithms gave the same results. Similar results were also observed for the students' year of study where students at their third and fourth years of study had higher cumulative GPAs than those in their first and second years of study.

However, similar tests showed that the English and Information Systems Technology (IST) placement scores had less impact on the students' cumulative GPA. This showed that students do not require prior knowledge in computers to do well in computer-related degree programs. Prior knowledge in computers refers to computer literacy skills. The cluster centers of high cumulative GPA scores were close to the 50% test scores. However, further investigation to understand why factors such as previous computer knowledge were not influential contrary to popular belief is required. Similar results were also observed for the students' secondary school final exam GPA despite the expectation that students who did well in highschool would also do well at university. The secondary school final GPAs did not have a big impact on the cumulative GPA scores at university. This is expected given that the students selected for the program had similar highschool final GPA scores based on the pre-entry requirements.

Table 1. Impact of gender and degree program on cumulative GPA

	MALE	FEMALE	INFORM. SYST.	COMP. SCIENCE
Mean cumulative GPA	2.465	2.892	2.659	2.387
Total	673	185	535	323

Other factors examined, such as the students' gender, degree program (Computer Science or Information Systems), and country of origin, had almost no visible impact at all when a cluster analysis was done. These findings concur with [25]. However, further investigation to understand why factors such as gender were not influential contrary to popular belief is required. The Independent Samples T-test [12] was used to compare the difference in cumulative GPA means where two independent groups existed such as male and female students, or Computer and Information Systems degrees. The purpose of the T-test was to determine whether the difference between the two groups was statistically significant. A T-test conducted on the students' cumulative GPA versus students' gender showed that the difference in cumulative GPAs was not statistically significant between the two groups as tabulated in Fig. 2a. The results— $t(323.035) = -7.300$ with $p = 0.000$ —show that there is no statistically significant difference in the variances between the two groups. The p -value is below the critical significance level of 0.05. Conducting a similar T-test on the students' degree program versus their cumulative GPA also confirmed that there was no statistically significant difference in the GPA reported between the two groups: Fig. 2b. The results— $t(664.239) = 5.017$ with $p = 0.000$ —show that there is no statistically significant difference in the variances between the two groups. The p -value is below the critical significance level of 0.05. The results statistically confirmed that gender and the degree program had no significant influence on the students' cumulative GPA.

It is worth noting that further analysis showed that there were more male students admitted in the two programs, although female students had comparatively higher cumulative GPAs as illustrated in Table 1a. The difference, however, was not statistically significant. Both computer-related degree programs had low female enrolments. Additionally, students in the Information Systems program were more than those in the Computer Science program. This was expected because the former program had been offered for a longer period. However, we observed that the Computer Science program had students with slightly higher cumulative GPA than those from the Information Systems program as illustrated in Table 1b, although the difference was also not statistically significant.

To assess the significance of each of these factors a Pearson Correlation analysis [23] was conducted. The bivariate Pearson Correlation produces a sample correlation coefficient, r , which measures the strength and direction of linear relationships between pairs of continuous variables. The Pearson Correlation was used to present the statistical evidence for variables that impact the cumulative GPA or not. The results of the correlation confirmed the observations made from

the clustering tests that mathematics placement test scores influenced the students' cumulative GPA. There is a statistically significant correlation between Mathematics placement test scores and a student's cumulative GPA. The Pearson's r is positive, indicating that when one variable increases the second variable also increases. The Sig (2-Tailed) value is less than or equal to 0.05, showing that the relationship is statistically significant. However, the correlation between a student's IST placement test score and a student's cumulative GPA was not statistically significant, because the value of the Pearson's coefficient r is close to zero. The correlation between a student's country of origin and a student's cumulative GPA was not statistically significant.

The results obtained so far identified variables that were significant in determining a student's cumulative GPA and the extent to which each variable impacted the cumulative GPA. They, however, do not describe patterns in the data showing how all the variables jointly impacted the cumulative GPA. This is useful when modeling relationships between variables. In order to describe patterns in the data, further analysis of all the variables was required to show how they influence each other. For this task, clustering and decision tree algorithms were applied to the data. When a cluster analysis was conducted with the EM clustering algorithm, three large clusters were identified. The results presented in Fig. 3 show three significant clusters labelled 2 (GPA-2.6), 6 (GPA-2.5), and 8 (GPA-2.1) as having the highest cluster densities, i.e.: 22%, 25%, and 22%. These clusters represent groups of students with similar characteristics. The math placement scores for each cluster were observed to be 59%, 53%, and 70% (Fig. 3). The high densities showed that majority of the students had these characteristics. Figure 3 further shows two clusters labeled 1 and 3 with high cumulative GPA scores of 2.9 and 3.1. The Mathematics placement scores are also higher at 71% and 68%. This confirms the earlier results that math placement score influenced the cumulative GPA. Although clustering showed the existence of relationships, it is difficult to tell at a glance what the relationships were. Additional analysis was required to describe the factors that formed each cluster.

Running decision tree algorithms on the data revealed the relationship between different variables as illustrated in Fig. 4. From the results it can be observed that the Introduction to Information Systems course (IST1020), the year of study (cl), and the math placement test played the most significant role in determining a student's cumulative GPA. The Introduction to Information Systems course imparts general computer literacy skills to students. It does not include any computer programming. A few second level courses were also found to have an impact, namely Computer Organization (taken by the Information Systems degree students) and Computer Networks (taken by the Computer Science degree students). The Computer Organization course exposed students to computer architecture and assembly language while the Computer Networks course introduced students to data communication protocols and devices. Both courses were taken by students as soon as they completed the introduction to Information Systems course. The higher-level courses had no impact. The find-

Clusterer output

Number of clusters selected by cross validation: 10
 Number of iterations performed: 1

Attribute	Cluster									
	0 (0.08)	1 (0)	2 (0.01)	3 (0.05)	4 (0.19)	5 (0.25)	6 (0.06)	7 (0.13)	8 (0.07)	9 (0.17)
<hr/>										
yr										
mean	2016.4927	2015.6489	2017.1951	2015.5709	2016.2478	2017.9099	2017.3803	2016.5287	2015.6185	2015.8332
std. dev.	0.9466	0.4989	0.8041	0.6883	0.6946	0.5293	0.9807	1.1246	0.7308	0.7685
deg_grant_yr										
mean	0	0	0	450.2433	0	0	0	0	12.1215	67.8244
std. dev.	255.9507	255.9507	255.9507	840.1341	0.0004	255.9507	255.9507	255.9507	155.9187	363.6887
cum_gpa										
mean	1.9424	2.9313	2.5933	3.0643	2.636	2.8615	2.5945	1.844	2.8628	2.5725
std. dev.	0.8623	0.5195	0.6426	0.5321	0.5567	0.6635	0.8868	0.8997	0.4782	0.539
math_placement										
mean	60.2561	71.9634	58.7228	68.3454	64.3582	68.7517	62.1589	52.6692	69.9384	61.7001
std. dev.	14.3854	4.6844	20.0351	13.3206	12.6818	12.0054	13.8341	18.0783	15.1788	14.3596
ist_placement										
mean	68.7286	75.2114	67.4928	62.7951	65.0351	74.5555	67.7237	61.4311	75.4127	65.2103
std. dev.	15.3161	8.4381	10.2259	28.5914	24.6406	7.5185	11.1584	17.5195	10.8344	21.6194

Fig. 3. Mining patterns through clustering

ings brought out the importance of laying a strong foundation for students during their initial courses in computing. The approach to the foundation classes served to either propel students to success or failure. It was observed that seasoned instructors who were professors in the department taught higher-level courses or graduate programs while early career instructors and part-time instructors were left to handle introductory level courses in the department. This approach needed to change to enhance students' performance.

Our study also sought to establish how courses taught at various levels influenced each other. To achieve this a Pearson Correlation test was done across the various courses offered. The courses are mapped against each other and the Pearson Correlation values are provided. The lower-level courses are courses offered in year 1 and year 2, indicated in green in the column and row headings in Fig. 5. The higher-level courses are courses offered in years 3 and 4, indicated in blue in the column and row headings. The results presented by the correlation matrix in Fig. 5(a) showed that for the Computer Science program lower level courses appeared to influence each other positively (Pearson Correlation value above 0.4), such that a high grade in one would most likely mean a high grade in the other. A closer examination revealed that this occurred between related courses such as Computer Organizations and Operating Systems or Web Design and Computer Programming. Related courses in this context are considered as courses that are a pre-requisite of the other. For example, Computer Organizations is a pre-requisite course for Operating systems and Computer Programming is a pre-requisite for Web Design. Unexpected relationships identified were between Computer Organization and Computer Programming which

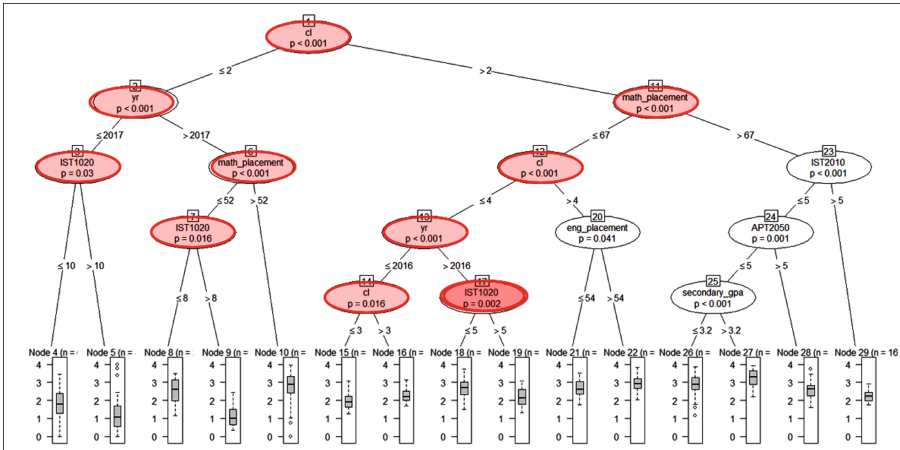


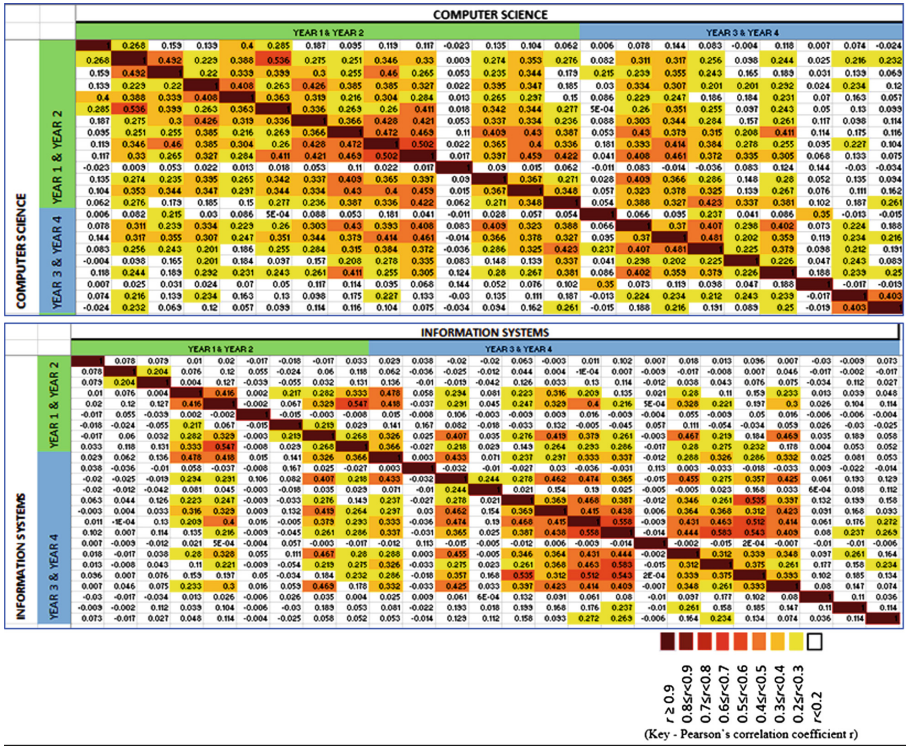
Fig. 4. Decision tree classification of factors influencing the cumulative GPA

are courses that are not indicated as prerequisites of each other. This, however, might be attributed to the fact that Assembly Language Programming was part of the Computer Organization course.

For the Information Systems program, fewer courses seemed to influence each other, as depicted by the presence of more white cells in Fig. 5b (Pearson Correlation value less than 0.2). This was a possible indication of the lack of proper linking when teaching courses across the curriculum. Some significant relationships, however, were found among related higher-level courses such as System Analysis and Design, Object Oriented Programming, and Database Management Systems, as well as Decision Analysis, Data Structures and Algorithms, and Mobile Programming. These relationships were expected because the courses were related and, therefore, had been stated as pre-requisites of each other in the curriculum. One unexpected relationship, however, was identified between the Digital Laboratory and Object-Oriented Programming where students who did well in Digital Laboratory also did well in Object-Oriented Programming and vice versa. The two courses had distinct content and, therefore, had not been indicated as pre-requisites of each other in the curriculum. Further research is required to investigate the cause of this relationship.

4.2 Strategy for Enhancing Performance

The experiments described in the previous sub-section provided insights into weaknesses in the current degree programs offered at our university. Our results can thus provide an opportunity to review the programs and to align courses properly to enhance students' academic performance in computer-related programs. Strategies on the courses to be considered when admitting students into the program, the courses to be emphasized or restructured in the curriculum, and ways of reducing attrition rates through performance are discussed in the following paragraphs.



on how the courses are taught to develop students' interest as well as to lay a solid foundation for our two computing programs. Basic skills that will be required in advanced courses should be instilled at the foundation level. Further research on the specific skills required at this level is required. Such skills should, therefore, be integrated into the Introduction to Information Systems course. Senior and experienced instructors, who traditionally teach higher level courses and (post)graduate courses, should be encouraged to participate in lower-level courses to prepare and mentor the students well during their foundation phase.

Structure of the Curriculum. Our results further show that there was a significant positive correlation among courses in our Computer Science degree. Students who did well in the introductory programming courses also did well in the advanced programming course. This was, however, not the case in our Information Systems degree program. This was attributed to the fact that our Computer Science curriculum was developed after the Information Systems degree and, therefore, structured already better to accommodate 'lessons learned' from the Information Systems degree. The structure of the Information Systems degree program has since been revised to address some of these challenges. It is envisaged that the effects of these changes will appear in future studies.

Linking Courses Across the Curriculum. We also noted the need to relate courses when teaching the program as well as re-considering prerequisites within the program. While relationships were observed among programming courses, the same was not apparent among networking courses or computer security courses. Students who did well in the lower-level networking and security courses should also be able to do well in similar courses at the advanced level. This absence of well-defined module relationships could, however, be due to the absence of courses at a higher level to continue their matching lower-level courses in those tracks. Further research is required to establish the reason for the lack of such relationships.

Prediction Modeling for an Early Warning System. Machine learning and data mining techniques can be combined to identify weaknesses in a degree program that may not be easy to detect without such tools. Through prediction modeling, forecasts on students' performance can enable stakeholders to proactively take measures to enhance students' performance. Automation of the process also eliminates human biases that can interfere with the process. It provides an opportunity for instructors to identify with minimal effort weaknesses in the programs offered. Through the provision of early warnings, student attrition rates can be reduced and subsequently the quality of education can be improved.

5 Limitations

Our study investigated the academic performance of students taking computer-related degree programs at university. The general objective of this study was

finding out whether factors influencing students' academic performance can be modeled to provide explanations for students' performance with a view of enhancing their performance. Although the study was conducted in a single private university, the results have the potential to be generalized for other institutions. Further research in this area should include a comparative study among several institutions to provide recommendations for enhancing our approach.

Traditionally many universities emphasize that mathematics and physics skills are required for admission into computer-related degree programs. Our findings confirmed that background skills in mathematics were a key factor in determining a student's performance. Our results provide a basis for investigating students' performance. Further research is required to describe the correlations with a view of establishing whether they are pointers to actual correlations or merely spurious. Further, we did not investigate the relationship between the students' background knowledge in physics and their academic performance. This is because physics knowledge was not assessed by the university, and no records of the students' secondary school scores for physics were captured in the university's Student Information System.

Students' academic performance is influenced by several factors originating in their social, economic, and academic backgrounds. Our research, however, only focused on academic factors arising from students' educational backgrounds as well as their current academic performance in courses taken at our university. These were factors that could easily be collected from the Student Information System. Additionally, they are factors that the institution itself can address to enhance the students' academic performance. Our study revealed the *need to improve the teaching approaches* for lower-level courses as they have an impact on the students' cumulative GPA later in the program. However, we did not investigate how to do this. Further research is required to explore how different teaching approaches influence the cumulative GPA.

6 Conclusions and Future Research Directions

In our study we carried out explanatory modeling to reveal some of the factors that influence students' academic performance in computing. The suggested strategies are based on anecdotal evidence. Nonetheless they provide a basis for further research to investigate the effectiveness of the suggested approaches.

A review of existing literature revealed that few similar studies had been done. Most of the research focused on enhancing the performance of specific courses in computing programs rather than all courses across the curriculum. Several studies used students' perceptions which is important but might not always yield the actual picture. Our study set out to address this gap by analyzing performance across all courses in a program with an emphasis on factors that the institution can influence. Further, we used actual student scores to evaluate factors that influence performance. Data analytics techniques were used to classify data and to provide explanations. The results obtained showed that students' mathematics background and their performance in their introductory computing

course were key in determining performance in their computer studies. Unexpectedly, prior computer skills or secondary school grades had less influence on a student's performance. The non-academic factors measured, such as country of origin and gender, were also not significant in determining a student's academic performance. We also suggested strategies for enhancing students' performance in computer-related degree programs.

Further research in this area should be conducted to validate the effectiveness of the suggested strategies. More extensive research should also be conducted among both private and public universities in the region as well as among other degree programs. Other factors that influence students' academic performance, such as their social, cultural, and economic backgrounds, would also enhance the findings of research in this area. Investigating the influence of other academic factors, such as students' background in physics, could further enhance the study. In our study, explanatory modeling laid the foundation for our current research in predictive modeling. Our goal was to show that explanatory modeling can be used to identify factors that influence performance—hence setting the case for predictive modeling of academic performance. The use of prediction models such as Regression Analysis and Neural Networks can provide a proactive approach to the problem. While our research is not exhaustive it presents an opportunity for other researchers in computer studies education to find ways of enhancing students' academic performance through explanatory and predictive modeling techniques.



References

1. Akinola, O.S., Nosiru, K.A.: Factors influencing students' performance in computer programming: a fuzzy set operations approach. *Int. J. Adv. Eng. Technol.* **7**(4), 1141–1149 (2014)
2. Al Murtadha, Y.M., Alhawiti, K.M., Elfaki, A.O., Abdalla, O.A.: Factors influencing academic achievement of undergraduate computing students. *Int. J. Comput. Appl.* **146**(3), 23–28 (2016)
3. Azcona, D., Smeaton, A.F.: Targeting at-risk students using engagement and effort predictors in an introductory computer programming course. In: Lavoué, É., Drachler, H., Verbert, K., Broisin, J., Pérez-Sanagustín, M. (eds.) *EC-TEL 2017. LNCS*, vol. 10474, pp. 361–366. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-66610-5_27
4. Barlow-Jones, G., van der Westhuizen, D.: Pre-entry attributes thought to influence the performance of students in computer programming. In: Liebenberg, J., Gruner, S. (eds.) *SACLA 2017. CCIS*, vol. 730, pp. 217–226. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-69670-6_15
5. Chikumba, P.A.: Student performance in computer studies in secondary schools in Malawi. In: Popescu-Zeletin, R., Rai, I.A., Jonas, K., Villafiorita, A. (eds.) *AFRICOMM 2010. LNICST*, vol. 64, pp. 113–121. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-23828-4_11
6. Garcia, R.A., Al-Safadi, L.A.: Comprehensive assessment on factors affecting students' performance in basic computer programming course towards the improvement of teaching techniques. *Int. J. Infonomics* **6**, 682–691 (2013)

7. García-Mateos, G., Fernández-Alemán, J.L.: A course on algorithms and data structures using on-line judging. *ACM SIGCSE Bull.* **41**(3), 45–49 (2009)
8. Giannakos, M.N., Pappas, I.O., Jaccheri, L., Sampson, D.G.: Understanding student retention in computer science education: the role of environment, gains, barriers, and usefulness. *Educ. Inf. Technol.* **22**(5), 2365–2382 (2017)
9. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H.: The WEKA data mining software: an update. *ACM SIGKDD Explor. Newsl.* **11**(1), 10–18 (2009)
10. Kim, H., Song, H.Y.: Formulating human mobility model in a form of continuous time Markov chain. *Proc. Comput. Sci.* **10**, 389–396 (2012)
11. Khomokhoana, P.J., Nel, L.: Decoding source code comprehension: bottlenecks experienced by senior computer science students. In: Tait, B., et al. (eds.) *SACLA 2019*. CCIS, vol. 1136, pp. 17–32. Springer, Cham (2019)
12. Kim, T.K.: T-test as a parametric statistic. *Korean J. Anesth.* **68**(6), 540 (2015)
13. Kumar, A.N.: Need to consider variations within demographic groups when evaluating educational interventions. *ACM SIGCSE Bull.* **41**(3), 176–180 (2009)
14. Kremelberg, D.: *Practical Statistics: A Quick and Easy Guide to IBM SPSS Statistics, STATA, and other Statistical Software*. SAGE, Thousand Oaks (2010)
15. MacQueen, J.: Some methods for classification and analysis of multivariate observations. In: *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, pp. 281–297 (1967)
16. Namuye, S., Platz, M., Okanda, P., Mutanu, L.: Leveraging health through early warning systems using mobile and service-oriented technology. In: *Proceedings of IEEE IST-Africa Conference*, pp. 1–10 (2015)
17. Nash, J.: Computer skills of first-year students at a South African University. In: *Proceedings of SACLA 2009 Annual Conference of the Southern African Computer Lecturers' Association*, pp. 88–92 (2009)
18. Njoroge, M.M., Wang'eri, T., Gichure, C.: Examination repeats, semester deferrals and dropping out as contributors of attrition rates in private universities in Nairobi County Kenya. *Int. J. Educ. Res.* **4**(3), 225–237 (2016)
19. Odhiambo, G.O.: Higher education quality in Kenya: a critical reflection of key challenges. *Qual. High. Educ.* **17**(3), 299–315 (2011)
20. Pretorius, H.W., Hattingh, M.J.: Factors influencing poor performance in systems analysis and design: student reflections. In: Liebenberg, J., Gruner, S. (eds.) *SACLA 2017*. CCIS, vol. 730, pp. 251–264. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-69670-6_18
21. R Core Team: *R: a Language and Environment for Statistical Computing* (2013). <https://cran.r-project.org/doc/manuals/fullrefman.pdf>
22. Rauchas, S., Rosman, B., Konidaris, G., Sanders, I.: Language performance at high school and success in first year computer science. *ACM SIGCSE Bull.* **38**(1), 398–402 (2006)
23. Sedgwick, P.: Pearson's correlation coefficient. *BMJ* **345**, e4483–e4483 (2012)
24. Wang, X.M., Hwang, G.J., Liang, Z.Y., Wang, H.Y.: Enhancing students' computer programming performances, critical thinking awareness and attitudes towards programming: an online peer-assessment attempt. *J. Educ. Technol. Soc.* **20**(4), 58–68 (2017)
25. Wilson, B.C.: A study of factors promoting success in computer science including gender differences. *Comput. Sci. Educ.* **12**(1/2), 141–164 (2002)



The Use of Industry Advisory Boards at Higher Education Institutions in Southern Africa

Estelle Taylor¹✉  and Andre P. Calitz² 

¹ Department of Computer Science and Information Systems, North-West University,
Potchefstroom, South Africa
estelle.taylor@nwu.ac.za

² Department of Computing Sciences, Nelson Mandela University,
Port Elizabeth, South Africa
andre.calitz@mandela.ac.za

Abstract. An Industry Advisory Board (IAB) can provide useful feedback to academic schools or departments, relating to topics such as industry graduate requirements, IT trends, programme quality and curriculum development. Though the existing literature already provides general guidelines for the role and responsibilities, membership, composition and functioning of IABs, literature on the use of IABs specifically in southern Africa is limited, especially as far as best practices and perspectives for the use of IABs for Computer Science (CS), Information Systems (IS) or related IT departments (IT) are concerned. Hence, the question addressed in this paper is: How are IABs used by CS/IS/IT/ICT departments in higher education in Southern Africa? An IAB questionnaire was compiled and sent to the Heads of Departments (HODs) of 32 universities in southern Africa. Accordingly, feedback received from IABs could have a direct impact on the ICT curricula and could also assist ICT lecturers in their efforts to update relevant course contents in a continuously changing computing environment. This paper might also help academic CS/IS/IT/ICT departments to implement and maintain IABs and to follow the standards of best practice.

Keywords: Industry advisory boards · ICT curriculum development · Quality management

1 Introduction, Research Problem, and Research Design

Academic departments in Computer Science (CS), Information Systems (IS), Information Technology (IT) and other related departments (subsequently also abbreviated collectively as ICT) at Higher Education Institutions (HEI) should offer programmes and curricula that are relevant in a fast-changing environment, where students should be prepared for a continuously changing workplace¹.

¹ For comparison see parts II-III of [17].

Organisations like the IEEE Computer Society and the ACM have made efforts to specify requirements or content for programmes in CS (CS2016), IS (IS2010) and IT (IT2017), though there is no clear specification on the graduate attributes required by the ICT industry [19]. Communication and collaboration between academics and stakeholders from industry therefore should play an important part in this effort. *“Today, more than ever, a powerful and influential advisory board is essential for the success of an educational institution”* [24].

Maintaining contact with Industry Advisory Board (IAB) members and obtaining feedback from the IAB members on the quality of academic programmes have become an important activity at ICT departments at HEIs and comprehensive universities offering vocational diplomas and academic degree programmes [13]. IABs provide perspective and are a valuable source of advice about matters such as curricular issues, career choices and guidance for academic department members and students [15, 22, 24]. According to the literature, departments may use IABs to provide diverse perspectives, to gain input or advise to customise or strengthen programmes [21, 24] or programme quality [10], and to uniquely shape course content [19]. Additionally, IABs are used to monitor the effectiveness of curricula or the performance of graduates, to enable academic departments to keep in touch with the trends and the needs of the industry, or to help academic departments to meet accreditation requirements [21]. Thus the feedback obtained from IABs should be used by management as well as lecturers to keep ICT curricula and module contents up-to-date and relevant.

Accreditation boards in most cases require the existence and use of IABs by academic schools or departments. Even though it is clear from the above that having and using an IAB is to the advantage of HEIs, there is not sufficient data available on whether all ICT departments at HEIs in southern Africa have IABs, or—if they do—whether they use their IABs in the most effective way. Departments that do not have any IAB ought to profit from guidelines for the establishment and use of IABs. Departments that have an IAB ought to use standard practices and guidelines to ensure that they use their IAB in the most effective way.

Zahra (et al.) referred to advisory boards as an under-researched contributor to education and indicated that it has become essential to gather more data about these boards, as academic literature on the role and composition of advisory boards is sparse [25]. Therefore, this paper describes the current practices by ICT departments in southern Africa on the use of IABs as well as the roles and responsibilities of the IAB members and topics relating to meetings. Our research problem, research questions and the IAB survey are outlined in the subsequent paragraphs. Related work on IAB composition, meetings, roles and responsibilities is recapitulated in Sect. 2. Our IAB survey results are presented in Sect. 3. Our conclusions and recommendations (relevant to ICT departments using IABs) as well as future work are described in Sect. 4.

Problem Description. Söderlund (et al.) warned that a *recent decline in scholarly interest in advisory boards* may limit the understanding of the advantages and complications of IABs [23]. They further refer to a *lack of studies* on best practices and perspectives relating to the use of IABs. According to [25], further research on IABs would help departments to better understand the role and attributes of advisory boards, as well as how to best realise their potential. Management, as well as lecturers, should be informed of the importance of feedback from industry IAB members, especially in the fast-changing field of ICT. Lecturers must deliberate the feedback obtained from an IAB and evaluate the impact it may have on the curriculum and module contents [14,24]. Thereby it is important that lecturers are involved in the process, as they are required to consider the feedback from the IAB and to make changes to the courses they teach. These changes could lead to the institution delivering graduates who were taught an improved, relevant programme in the field of ICT [9].

Literature differs on the factors of effective and efficient IABs, such as size, members, number of meetings per year, and the like [11]. The problem investigated for this paper is based on the recognition that there is insufficient recent literature on the use of IABs by HEIs in southern Africa. Only a small number of CS and IS curricula are currently accredited in the Republic of South Africa, whereby their HEIs must comply with an accreditation body's IAB requirements. Further, there is also a lack of guidelines for academic ICT departments on the best use of IABs in southern Africa. Therefore, our paper addresses the following

Research question: How are IABs used by ICT departments at HEIs in southern Africa?

The aim of our study is to investigate the use, functioning and practices used by ICT schools and departments at HEIs in southern Africa that already appoint an IAB.

Method. An IAB questionnaire was compiled on the basis of similar earlier studies [7,10]. In order to obtain honest personal perceptions, we decided to keep the survey anonymous. Our IAB questionnaire consisted of the following sections:

- Biographical details;
- Twelve open-ended questions relating to the use of an IAB by the specific department/school; and
- Likert scale questions relating to IAB membership, IAB meetings and documentation, roles and responsibilities, and functioning.

The questionnaire was captured with the Nelson Mandela University's online survey tool, QuestionPro. Thereafter we contacted the heads of ICT departments (HODs) from all 32 universities in southern Africa. The first call for participation was distributed via e-mail to the HODs listed in the SACLA HOD list. A second call was sent to various HODs individually. 23 HODs (or deputies)

completed our questionnaire during a two-week period. Their answers were statistically analysed with the help of the Statistica tool, whereby the non-numeric (qualitative) results were thematically analysed with AtlasTi.

2 Related Work

2.1 Definition and Role of Industry Advisory Boards

A paper entitled “*Reversing the Landslide in Computer-Related Degree Programmes*” [4] suggested that the introduction of an IAB should be the first response to declining student numbers. Accreditation of programmes should be shaped in a process that involves alumni, IABs, faculty members and students [19]. According [14], advisory boards in the IS environment date back to the 1960s and can be defined as a *group of qualified volunteers whose goals include providing direction, content and resources, professional development, research, curriculum, resources and strategic direction*. An advisory board can also be defined as a *group of professionals brought together with the aim of helping an academic centre to accomplish its mission* [25]. In this paper, the term ‘IAB’ has the following

Definition: An IAB is an independent body established by an academic department in order to enable it to form and implement its vision and mission.

Universities provide general guidelines and policies regarding the functioning of an academic department’s IAB [6]. For example, the objectives of an IAB indicated in the Nelson Mandela University’s IAB Policy include:

- Building relationships between relevant businesses and the department;
- Providing guidance and giving advice on, for example, strategic direction with regard to trends, professional, business and management practice;
- Providing practical requirements; and
- Providing advice on graduate skill requirements.

Other literature sources indicate as objectives or purposes of IABs:

- Research and programmes [14];
- To act as advocate for the programme [23];
- To provide guidance on academic issues and planning [8];
- To advise on matters relating to new degree programmes and options, long term planning, community relations, development and policy matters [8, 23, 24];
- To operate at the strategic level in analysing industry needs and trends, as well as in reviewing and monitoring programme objectives [12];
- To monitor learning and research quality and impact, as well as the progress in fulfilling mission and plans [25];
- To act as a link between an academic department and its industrial and professional partners [8];
- To mentor students [23];
- To assess the quality and skill of graduates [21]; and
- To assist with internships (students) and job placement (graduates) [23];

The important role of IABs is highlighted in [16]: according to this study, 46% of the departments of ICT (from 23 HEI in South Africa) used IABs at that time.

2.2 Different Requirements: Members, Size, and Meetings

The selection of members for an IAB is important. The Institute for IT Professionals in New Zealand indicate that IAB members can assist with developing the body of knowledge in an academic programme as they are potential employers of the programme graduates [12]. According to [24], members should be selected from business, industry, government with variety. El Refae (et al.) additionally mention alumni, students, leaders of the profession, different genders, and ethnic minorities [10].

Literature sources differ on factors such as size, members, number of meetings per year, and the like, of IABs. If a board is too small, it lacks critical mass, but if it is too large it is difficult to take everyone's opinion into account when dealing with complex issues. Smaller boards are more efficient in decision-making, whereas larger boards provide more resources [14]. According to [21], the board should consist of 20 to 25 individuals. Mandviwalla (et al.) refer to four separate cases in which the boards consisted of 17 members ($\approx 44\%$ alumni), 35 members ($\approx 10\%$ alumni), 60 members ($\approx 15\%$ alumni) and 11 members ($\approx 50\%$ alumni) [14].

In another case the advisory board consisted of 17 members (6 faculty members, 1 student representative and 10 external advisers) [9]. In [25], 31 advisory boards were studied: they had between 7 and 16 members. In [23] with 10 cases, the number of members varied from 4 to over 20 from case to case. In general, literature sources indicate a minimum of 10 members with an average of 20 members [11, 18]. Affiliations of IAB members varied between exclusively from industry, a mixture from industry and academia, as well as from industry, faculty, plus student representatives.

Leadership of the IAB differs from, for example, an externally appointed chair to an internally appointed chair [14]. Members can be chosen from companies which frequently hire graduates from the specific institutions [21]. In other cases the membership choice is focused on senior executives [14].

The scheduling of meetings also differs, for example twice per year [21], or three 2.5-h evening meetings per year, two 1.5-day meetings plus an annual meeting with a director and sub-committee meetings, or two half-day meetings [14]. In [25], the average for 31 advisory boards was 4–6 meetings a year with each meeting lasting about 2 h. In the 10 cases of [23], the frequency of meetings varied from 1 to 6–12 per year.

The purpose of IAB meetings can be to discuss pending issues (for example the content and breadth of the curriculum) and to 'brainstorm', or to 'network' and report changes [21]. In addition, a meeting should ideally generate significant input to establishing strategies for monitoring the development of technical competence as well as personal and professional skills for each particular programme [12].

2.3 Benefits and Challenges of IABs

The use of an IAB can be to the benefit of the industry, the academic institutions, and their students.

Benefits for the Industry include:

- Interaction with faculty members during meetings, sustaining strategic relationships [14,24];
- Influence on academic curricula, programmes and research towards industrial needs [9,14,24];
- Interaction with other industry representatives [9,14,24];
- Access to faculty members for short further-education courses [24];
- Development of talent [9,14];
- Access to competent graduates for employment [6];
- Development of insights [14];
- Social prestige by ‘giving back’ to higher learning in general and by fulfilling societal responsibilities [9,14,23].

Benefits for the Academic Departments include:

- Latest industrial ‘best practice’ can inform research and curriculum, which in turn entails that the programmes can remain at the forefront of technological innovation [8,9,14,25];
- Improved curricula [8,9,14,25];
- Improved programme quality assurance [10];
- Validation of direction and plans, as well as recommendation of initiatives [8,14,24];
- Higher visibility and better public relations as well as funding or donations of equipment [8,9,14,24,25];
- Increasing student-enrolment [24];
- Contact networks, resources and data for research [14,25];
- Increased reputation [14];
- Increased influence beyond the institution [14];
- Connection to various influential stakeholders [25].

Benefits for the Students include:

- A timely curriculum that prepares them well for employment [9];
- Mentorship obtained from board members [9,14];
- Job placement and recruitment opportunities [3,9];
- Interesting guest lectures given by industrial speakers [3];
- Industry visits and internships [7].

Challenges. Söderlund (et al.) described challenges experienced by various institutions with the use of advisory boards [23]. Mentioned are the ‘logistics’ of setting up the advisory boards, facilitating meetings, scheduling and conducting meetings, the intensity of meetings becoming tedious for the stakeholders, overwhelming amounts of advice that must be acted upon, as well as differences in personality and/or opinions.

2.4 Accreditation Board Requirements for Industry Advisory Boards

According to the Technology Accreditation Commission (TAC) of the US Accreditation Board for Engineering and Technology (ABET), accredited programmes must have an industrial advisory committee including industry representatives [1]. Accreditation counts as proof that a programme meets the essential standards to produce graduates to enter the critical professions of science, technology, engineering, and mathematics (STEM). Accreditation bodies like the British Computer Society (BCS) [5], the Accreditation Board of the Australian Computer Society [2], or the IT Professionals New Zealand [12] operate the IT industries' degree accreditation processes, endorse degree programmes, and support the international portability of qualifications. The use of IABs is endorsed by these organisations.

SACAB, the newly established South African Computing Accreditation Board, consists of senior academic and industry representatives. It evaluates and endorses ICT degree programmes and diploma programmes at South African HEIs to meet international standards. It supports international portability of degree programmes and provides graduates with information about the career paths and skills required by the regional IT industry [20]. SACAB is currently in the process of establishing best practice guidelines for the functioning of IABs for academic ICT departments seeking accreditation of their programmes along the above-mentioned lines of international practice.

2.5 IAB Guidelines from the Literature

It is important that advisory boards make the best use of the valuable time industry and faculty members spend at meetings: *“If you want it to be valuable, you’ve got to put in the effort”* [23]. The following guidelines can be found in [14]:

- Mission and objective statements are essential for forming the direction and structuring conversations for the board. Keep the mission statement simple. This can include educational goals (curriculum design), research goals, funding and reputational activities²;
- Schedule regular meetings to maintain continuity;
- Establish dates a year in advance;
- Stick to the agenda;
- Give feedback on action items from previous meetings;
- Distribute the minutes soon after the meeting;
- Establish policies on attendance by conference calls, number of missed meetings and substitutes;
- Provide opportunities to socialise (see Footnote 2);
- Engage students, because interaction with students is an incentive for active participation in an advisory board (see Footnote 2).

² This aspect is also emphasised in [23].

Additional guidelines from other literature sources include:

- Exercise care in the selection of companies from which members are taken: The business relevance to the programme objectives must be considered [8];
- Choose members from companies that understand higher education [8];
- Find members from a diverse representation of the relevant profession [23];
- Start with a small board, which is more manageable, and expand as necessary or if advised to do so [23];
- Efficient planning of the agenda of meetings is very important, which also shows respect for members' time: Meetings should be focused on specific issues which are selected in advance [23];
- Supplement regular meetings with occasional e-mails and reports [23];
- Ensure that members understand exactly what is expected from them [23].

Conclusion. Our literature review of above has shown differences in opinions on issues surrounding IABs, including their objectives, membership, benefits, size, leadership, meetings and guidelines. In the following section the results from our own IAB survey in southern Africa are described.

3 Advisory Board Survey Results

Our *IAB* survey questionnaire was completed by representatives from universities in South Africa, Namibia and Zambia. 32 universities were initially 'listed' for the survey questionnaire which was eventually completed by 18 HODs and 5 HOD-representatives from 23 ICT departments at 17 universities. The respondents represented departments or schools of Computer Science (6), Information Systems or Informatics (9), combined CS&IS (3), other schools of ICT (2), and other similar departments (2). 44 persons initially 'viewed' the questionnaire (online), 32 started to provide answers, whilst 23 completed it. As the questionnaire had been initially emailed to 70 people on the SACLA HOD list, the response rate was 33%.

17 departments (74%) indicated that they had an IAB. 6 departments indicated that they had none. The departments without IAB indicated either that they were currently busy establishing a board, or had informal arrangements with companies, or that IABs would not serve departmental needs. No school or department compensated IAB members for serving on the board.

An analysis of the responses (Table 1) shows that the average size of an IAB is approximately 20 members. This includes academics and industry members. 9 departments indicated that they have on average 12 academics from their own school/department on the IAB. The difference in the number of members (1–50) correlates with our literature review of above³. 6 departments indicated that they have 2–3 students on their IABs, whereas another 6 departments indicated they do not include any student representatives.

³ More research might be needed on the optimal number of members.

4 departments indicated that they do not have alumni on their IABs. 6 departments indicated to have an average of 6 alumni on their IABs. 4 schools or departments include academics from other HEIs on their IABs. 12 departments indicated that they have had an IAB for an average of 12 years. 8 schools' or departments' IABs meet once per year; 4 meet twice per year. The average IAB meeting duration indicated by 11 departments was 3 h⁴.

Table 1. IAB membership

QUESTION	MEAN	MEDIAN	MIN.	MAX.
How many members are on your IAB?	19	17	1	50
How many departmental members serve on your IAB?	11	8	0	30
How many students serve on your IAB?	2	3	0	4
How many Alumni serve on your IAB?	4	3	0	19
How many years has your School/Department had a IAB?	11	9	5	20
How often does your IAB meet per annum?	1	1	1	2
What is the average duration (hours) of an IAB meeting?	3	3	1	5

The *professional roles* of the IAB members include senior IT managers, IT consultants, senior managers and IT managers from the manufacturing industry; one department mentioned managers from governmental entities (Fig. 1a). 4 IS departments indicated that they include senior managers from auditing firms on their IAB.

The *criteria* (Fig. 1b) used by schools or departments to select and appoint IAB members included: being senior managers in industry ($n = 6$), or IT specialists ($n = 5$), or alumni ($n = 3$), as well as knowledge and experience in the IT industry ($n = 3$). Also mentioned were *involvement with department*, *IT qualifications*, *employees of graduates*, as well as *availability* (in correlation with our literature review).

The *topics discussed* at IAB meetings mainly focus on the curriculum ($n = 7$), departmental activities ($n = 6$), teaching issues ($n = 5$), industry trends ($n = 4$), as well as departmental strategy, achievements and graduate employability (Fig. 2a). The teaching issues discussed most recently also included topics such as student protests and contingency planning. Additional were South Africa's changing education landscape, guest lectures, internships, project days and joint project benchmarking in comparison with other institutions.

⁴ For comparison: Most of the studies described in the literature also referred to 1–2 meetings per year, even though in [25] the average for 31 boards was 4–6 meetings per year.

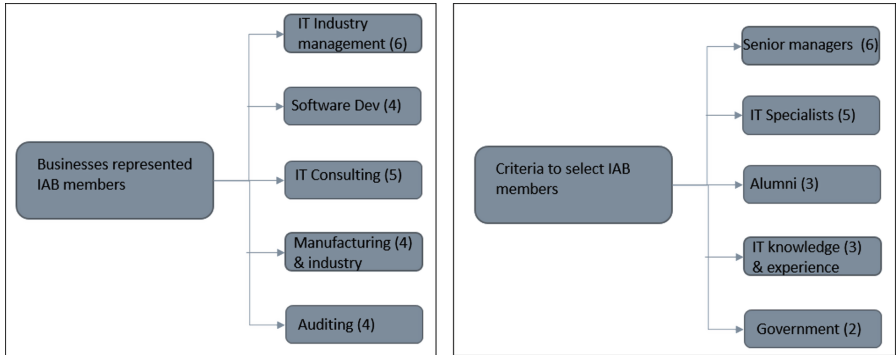


Fig. 1. (a) Businesses represented. (b) IAB member selection criteria

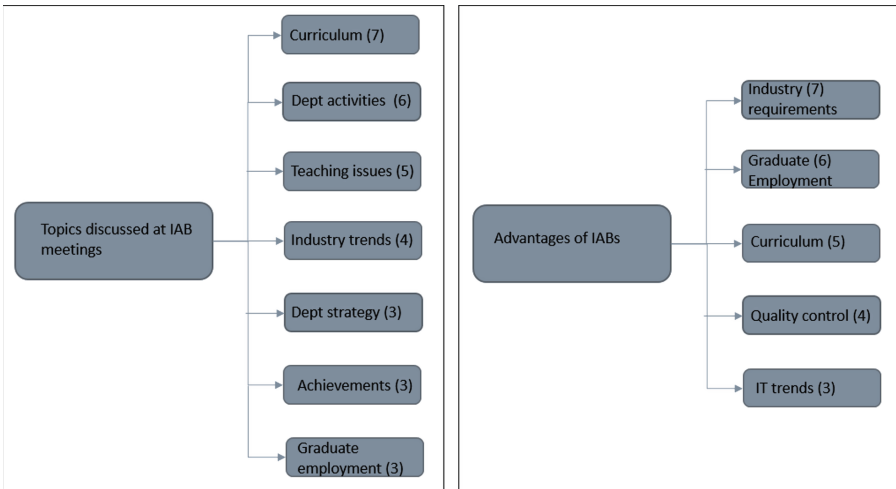


Fig. 2. (a) Topics discussed on IABs. (b) Advantages of IABs

Respondents indicated that the *advantages of having an IAB* (Fig. 2b) were to remain up-to-date with industry requirements ($n = 7$) and IT trends ($n = 3$), secure employment of graduates ($n = 6$), receiving industry advice w.r.t. the curriculum ($n = 5$), and quality control ($n = 4$). Respondents also emphasised the importance of maintaining partnerships with industry and of receiving industry feedback on the skills required of graduates.

The *challenges* faced by having an IAB included attendance ($n = 5$), recruiting members ($n = 4$), “to make sure they feel it is worth their time to attend the meeting”, “getting the same people from industry to consistently attend”, “distance to the meeting venue and conflicting dates with members’ calendars”. One respondent remarked: “We initially focused on national organisations, but found that it was challenging to get the members to regularly attend meetings,

and decided to shift our focus to local members”—for comparison see the literature review of above.

The main purpose of having an IAB was explained as follows by respondents:

- “Provide industry links and collaboration”
- “To keep up to date with the needs from industry and to have ambassadors for your department in industry”
- “Partnerships with industry”
- “Understanding industry requirements and trends”
- “Check/maintain quality”
- “Industry graduate requirements”
- “Provide working experience for students and student-corporate engagements”
- “Advising and coherence on curriculum issues”
- “Funding and advice on current IS/CS trends”.

Advice given by respondents to institutions wanting to establish an IAB included: “Choose members with correct qualifications”, “choose senior managers”, “choose alumni in senior positions”, “choose your industry members carefully: they must be executives who can make decisions”, “having an IAB is not negotiable”, “select senior IT people who have the time” as well as “set clear expectations and do proper planning”.

About 77% ($n = 12$) of the respondents were familiar with the IAB requirements specified by accreditation bodies like ABET, BCS, or SACAB. One respondent indicated specifically that having the IAB was “because of accreditation requirements”. However the universities *outside* South Africa were not familiar with the IAB requirements specified by international accreditation bodies.

The 18 responses received on Likert scale statements regarding IAB membership, meetings and documentation, roles and responsibilities, as well as having success with IABs are presented below. In the subsequent discussion, the Likert scale categories ‘Strongly Disagree’ and ‘Disagree’, resp. ‘Strongly Agree’ and ‘Agree’, are combined for the sake of simplicity.

The majority of the respondents ($\approx 83\%$, $n = 15$) indicated that members of the IAB must have a relevant IT qualification (Fig. 3). About half of the respondents ($n = 9$) disagreed that members of the IAB must have studied at their institution; and only 6 departments or schools agreed. About 78% ($n = 14$) agreed that the IAB must include alumni, and all participants agreed that the IAB must be aware of the latest IT trends. The majority of respondents ($n = 12$) agreed that members of the IAB must be aware of the latest curriculum trends, and all agreed that members of the IAB must be active in the IT industry.

7 departments or schools indicated that the IAB must meet once per semester. Only 4 departments or schools indicated that the IAB minutes must be available on the departmental website (Fig. 4). 7 departments indicated that the IAB members’ names and documentation must be available on the departmental website. All respondents agreed that the IAB members must be made aware of their roles and responsibilities. The majority ($n = 16$) of the respondents agreed that the IAB agenda must include a discussion of new IT trends.

13 departments indicated that the agenda must include an item on new curriculum developments. Reporting on graduate employment figures is a requirement of accreditation bodies, and 12 departments agreed to include an item on the employment figures of graduates.

The majority of the respondents from universities in South Africa agreed that IAB members must be familiar with the programmes offered in their schools or departments. About 61% ($n = 11$) agreed that IAB members must have knowledge about where graduates are employed after completion of their studies. About 67% ($n = 12$) indicated that IAB members must have knowledge about the job opportunities available for graduates (Fig. 5). The majority ($\approx 79\%$, $n = 14$) indicated that the IAB members must have knowledge about the departmental staffing situation and requirements. All agreed that the IAB members must be aware of the departmental strategy, mission and vision. 7 departments indicated that the IAB must monitor the annual alumni survey conducted by the schools or department.

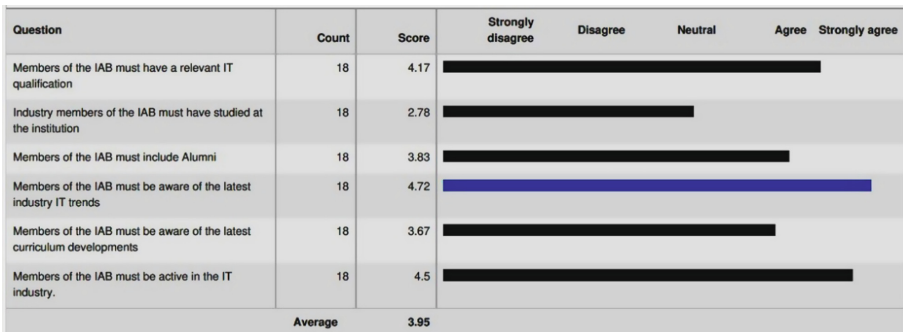


Fig. 3. IAB membership

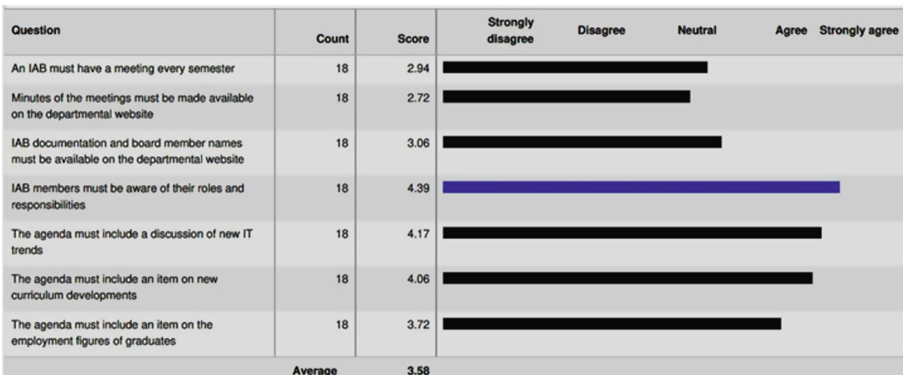


Fig. 4. IAB meetings and documentation

About 61% ($n = 11$) of the respondents indicated that the IAB assists in maintaining academic standards (Fig. 6). All participants agreed that the IAB provides a link to the industry. About 72% ($n = 13$) agreed that IABs assist with quality assurance for a department or school. A ‘mixed’ response—with $\approx 44\%$ ($n = 8$) agreement and $\approx 33\%$ ($n = 6$) disagreement—indicated that IABs might possibly highlight issues faced by a department or school with university management. The majority agreed ($\approx 89\%$, $n = 16$)—and nobody disagreed—that IABs can specify industry requirements, and ($\approx 89\%$, $n = 16$) that IABs can assist with curriculum and programme requirements. Finally, all respondents agreed that IABs can assist with the employability of graduates.

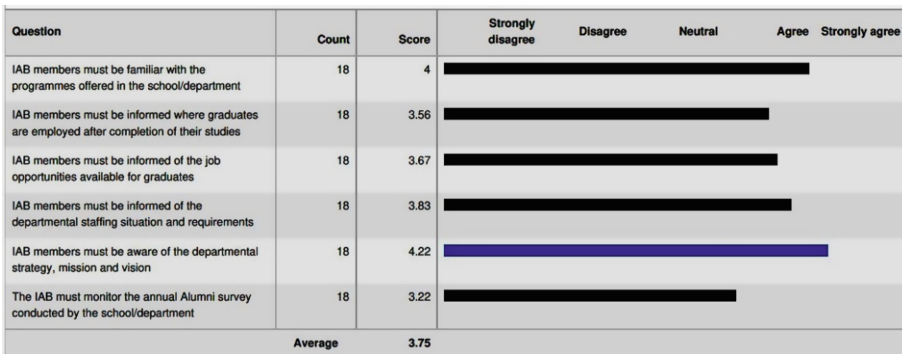


Fig. 5. IAB roles and responsibilities

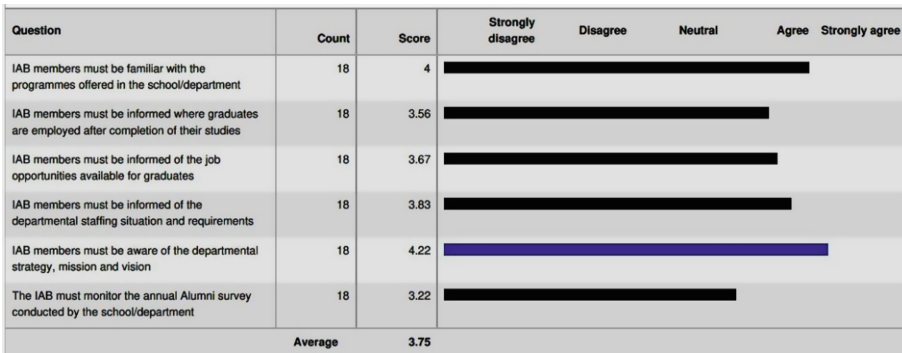


Fig. 6. IAB Successful IABs

4 Conclusions

Our literature study indicated that there are differences in opinions on issues surrounding IABs, such as the objectives, membership, benefits, size, leadership, meetings and guidelines. IABs have an important role for academic departments in maintaining academic standards and links to industry. Academics departments are required by accreditation bodies to have IABs.

Metkover and Murphy stated that $\approx 46\%$ of the departments of ICT have IABs [16]—compared to $\approx 74\%$ in our study. Only 6 out of 23 participating departments indicated that they did not have any IAB. The reasons mentioned were that they were establishing a board, or had informal arrangements with companies, or that the IAB was not serving departmental needs. Meetings are held 1–2 times per year whereby the duration of the average meeting is about 3 h. Topics discussed include the curriculum, activities, strategies, industry trends, graduate employability, and contingency planning.

Our analysis indicates that the average size of an IAB is about 20 members—including school or departmental members, alumni and other industry members, as well as one under-graduate and one post-graduate student representative. Thus our findings are similar to the ones of other studies which indicate a minimum of 10 members and an average of 20 members [11, 18]. The IABs include senior management members in the IT industry (e.g.: managers, consultants, and specialists). It is important that ICT lecturers are aware of the advantages of IABs, as this will encourage them to make themselves available for membership of IABs, and to network with the industry to enlist industry members for the IAB at their institution. The challenges mentioned by respondents were attendance (e.g.: conflicting dates on members' calendars and the distance to the venue), recruitment of members, and ensuring that it is worthwhile for members to attend.

Several advantages of having and maintaining an IAB were mentioned. These advantages included that IABs assist departments in keeping up to date with industry requirements and IT trends, secure employment of graduates, give advice on the curriculum and provide quality control. All of these advantages directly impact the lecturers as well as students. Lecturers in ICT face a constant challenge in keeping up to date with changes in the field and the modules they present. They themselves are, in most cases, not engaged with the industry. Lecturers are guided by IAB members on what they should be teaching to best prepare graduates to enter the workplace, whilst students receive the benefit of learning industry-relevant course content.

This first study on the use of IABs in southern Africa has provided a foundation for the continuous functioning of IABs as well as stakeholder management and engagement. Valuable opinions and information regarding IAB operations, procedures and composition were obtained from various ICT schools and departments in southern Africa. A list of guidelines for the use of IABs can now be formulated by combining the guidelines from previous literature with the guidelines emerging from our work. Such a list is currently being compiled. Future work shall include the development of a model together with guidelines for IABs'

effectiveness at HEIs in South Africa to be applied by the SACAB. More research can also be conducted on how the feedback from IABs ought to be used to update a curriculum, a module's contents, and—ultimately—the teaching and learning at HEIs.

References

1. ABET: Accreditation (2019). www.abet.org/accreditation/
2. ACS: Administration guidelines: Accreditation management manual (2016). www.acs.org.au/content/dam/acs/acs-documents/ACS-Accreditation-Documents-1-Administrative-Guidelines-V2.0.pdf
3. Albarody, T.M., Mohsen, A.S., Hussein, A.R., Melor, P.S., Yusoff, B.M.: Sustaining a synergistic industrial advisory board and academic collaboration. In: Proceedings of WEEF 7th IEEE World Engineering Education Forum, pp. 728–731 (2017)
4. Becerra-Fernandez, I., Elam, J., Clemmons, S.: Reversing the landslide in computer-related degree programs. *Commun. ACM* **53**(2), 127–133 (2010)
5. British Computer Society: Guidelines on Course Accreditation: Information for Universities and Colleges (2018). <https://www.bcs.org/media/1209/accreditation-guidelines.pdf>
6. Calitz, A.P.: A Model for the Alignment of ICT Education with Business ICT Skills Requirements. Doctoral Dissertation. (Business Adm.) Nelson Mandela Metropolitan University Business School, Port Elizabeth (2010)
7. Calitz, A.P., Greyling, J., Glaum, A.: CS and IS alumni post-graduate course and supervision perceptions. In: Gruner, S. (ed.) SACLA 2016. CCIS, vol. 642, pp. 115–122. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-47680-3_11
8. Craig, W.O.: Industry advisory board: a partnership between industry and academic department. In: Proc. Conf. for Industry and Educ. Collaboration, pp. 1–4 (2009)
9. Dorazio, P.: Professional advisory boards: fostering communication and collaboration between academe and industry. *Bus. Commun. Quart.* **59**(3), 98–104 (1996)
10. El-Refae, G.A., Askari, M.Y., Alnaji, L.: Does the industry advisory board enhance education quality? *Int. J. Econ. Bus. Res.* **12**(1), 32–43 (2016)
11. Genheimer, S.R., Shehab, R.: The effective industry advisory board in engineering education: a model and case study. In: Proceedings of 37th ASEE/IEEE Frontiers in Educational Conference (2007)
12. ITP New Zealand: Application Guidelines: ITP New Zealand Degree Accreditation (2017). <https://itp.nz/Activities/Degree-Accreditation>
13. Lending, D., Mathieu, R.G.: Workforce preparation and ABET assessment. In: Proceedings of the 2010 Special Interest Group on Management Information System's 48th Annual Conference on Computer Personnel Research on Computer Personnel Research, pp. 136–141 (2010)
14. Mandviwalla, M., Fadem, B., Goul, M., George, J.F., Hale, D.H.: Achieving academic-industry collaboration with departmental advisory boards. *MIS Quart. Exec.* **14**(1), 17–37 (2015)
15. Marshall, J.A.: Maximizing your industrial advisory board. *J. Ind. Technol.* **15**(2), 2–5 (1999)
16. Merkofer, P., Murphy, A.: The e-skills landscape in South Africa. *Zeitschr. für Politikberatung* **2**(4), 685–695 (2010)

17. Motta, G., Wu, B. (eds.): *Software Engineering Education for a Global E-Service Economy – State of the Art, Trends and Developments*. Springer, Cham (2014). <https://doi.org/10.1007/978-3-319-04217-6>
18. Pugh, S., Grove, M.J.: Establishing industrial advisory boards using a practice transfer model. *New Direct. Teach. Phys. Sci.* **10**, 20–25 (2014)
19. Reif, H.L., Mathieu, R.G.: Global trends in computing accreditation. *Computer* **42**(11), 102–104 (2009)
20. SACAB: South African Higher Education Computing Accreditation Board (2018). www.sacab.org.za/
21. Schuyler, P.R., Canistraro, H., Scotto, V.A.: Linking industry & academia: effective usage of industrial advisory boards. In: *Proceedings of American Society for Engineering Education Annual Conference Exposition*, vol. 6, pp. 1–5 (2001)
22. Sener, E.M.: Incorporating industrial advisory boards into the assessment process. In: *Proceedings of American Society for Engineering Education Annual Conference Exposition*, pp. 1–9 (1999)
23. Söderlund, L., Spartz, J., Weber, R.: Taken under advisement: perspectives on advisory boards from across technical communication. *IEEE Trans. Prof. Commun.* **60**(1), 76–96 (2017)
24. Summers, L.M.: Developing a college-industry relationship: the use of industrial advisory boards. In: *Proceedings of American Society for Engineering Education Annual Conference Exposition*, pp. 1–7 (2002)
25. Zahra, S.A., Newey, L.R., Shaver, J.M.: Academic advisory boards' contributions to education and learning: lessons from entrepreneurship centers. *Acad. Manag. Learn. Educ.* **10**(1), 113–129 (2011)

Author Index

- Ade-Ibijola, Abejide 3
- Bangani, Sifiso 35
Bradshaw, Karen 211
- Calitz, Andre P. 244
Chindeka, Vongai 211
- de Villiers, Carina 197
- Eybers, Sunet 181
- Futcher, Lynn 35, 50
- Gerber, Aurna 163
- Harneker, Roshan 64
Hattingh, Marie J. 181
- Jordaan, Joyce 197
- Kabaso, Sonny 3
Kafa, Violet 112
Khomokhoana, Pakiso J. 17
- Machoka, Philip 227
Mutanu, Leah 227
- Nel, Liezel 17
Ngwenya, Sandile 50
- Pilkington, Colin 131, 147
Pitso, Tshegofatso 197
Pretorius, Laurette 131
- Schütz-Schmuck, Marko 96
Siegburg, Marcellus 112
Smuts, Hanlie 163
Stander, Adrie 64
Steyn, Adriana A. 197
- Taylor, Estelle 244
- Uys, Walter F. 79
- van Biljon, Judy 147
van der Merwe, Alta 163
van der Merwe, Ronell 147
van Niekerk, Johan 35
Voigtländer, Janis 112