

**2024 36<sup>th</sup> International Conference on  
Software Engineering  
Education and Training (CSEE&T)**

**CSEE&T 2024**

**Editors: Andreas Bollin, Ivana Bosnić,  
Jennifer Brings, Marian Daun,  
and Meenakshi Manjunath**

**July 29<sup>th</sup> – August 1<sup>st</sup>, 2024**

**Würzburg, Germany**

Copyright © 2024 by The Institute of Electrical and Electronics Engineers, Inc.  
All rights reserved.

*Copyright and Reprint Permissions:* Abstracting is permitted with credit to the source. Libraries may photocopy beyond the limits of US copyright law, for private use of patrons, those articles in this volume that carry a code at the bottom of the first page, provided that the per-copy fee indicated in the code is paid through the Copyright Clearance Center, 222 Rosewood Drive, Danvers, MA 01923.

Other copying, reprint, or republication requests should be addressed to: IEEE Copyrights Manager, IEEE Service Center, 445 Hoes Lane, P.O. Box 133, Piscataway, NJ 08855-1331.

*The papers in this book comprise the proceedings of the meeting mentioned on the cover and title page. They reflect the authors' opinions and, in the interests of timely dissemination, are published as presented and without change. Their inclusion in this publication does not necessarily constitute endorsement by the editors, the IEEE Computer Society, or the Institute of Electrical and Electronics Engineers, Inc.*

ISBN-13: 979-8-3503-7897-9  
Online ISSN: 2377-570X

*Additional copies may be ordered from:*

IEEE Computer Society  
Customer Service Center  
10662 Los Vaqueros Circle  
P.O. Box 3014  
Los Alamitos, CA 90720-1314  
Tel: +1 800 272 6657  
Fax: +1 714 821 4641  
<http://computer.org/cspres>  
[csbooks@computer.org](mailto:csbooks@computer.org)

IEEE Service Center  
445 Hoes Lane  
P.O. Box 1331  
Piscataway, NJ 08855-1331  
Tel: +1 732 981 0060  
Fax: +1 732 981 9667  
<http://shop.ieee.org/store/>  
[customer-service@ieee.org](mailto:customer-service@ieee.org)

IEEE Computer Society  
Asia/Pacific Office  
Watanabe Bldg., 1-4-2  
Minami-Aoyama  
Minato-ku, Tokyo 107-0062  
JAPAN  
Tel: +81 3 3408 3118  
Fax: +81 3 3408 3553  
[tokyo.ofc@computer.org](mailto:tokyo.ofc@computer.org)

*Individual paper REPRINTS may be ordered at: <[reprints@computer.org](mailto:reprints@computer.org)>*



*IEEE Computer Society  
Conference Publishing Services (CPS)*  
<http://www.computer.org/cps>

# **36<sup>th</sup> International Conference on Software Engineering Education and Training (CSEE&T) CSEE&T 2024**

## **Table of Contents**

Message from General Chair .....	ix
Message from Program Chairs .....	xi
Organizing Committee .....	xiii
Program Committee .....	xv
Steering Committee .....	xix

### **Keynotes**

Mastering the Balance between Software Design Qualities .....	xxiii
<i>Sascha Kegreiß</i>	
AI and Software Engineering Education: Riding the Wave of Innovation .....	xxv
<i>Nancy R. Mead (Carnegie Mellon University)</i>	

### **ASEE&T Track**

Sustainable Competency-Oriented Computing Education for Π-Shape Elites and New Forms of Digital Education in the Future .....	xxix
<i>Xiaofei XU (Harbin Institute of Technology)</i>	
Constructive Alignment in Software Engineering Education .....	xxxii
<i>Axel Böttcher (Hochschule München University of Applied Sciences)</i>	
Competence-oriented Elaboration of Software Engineering Education .....	xxxiii
<i>Dieter Landes (Coburg University of Applied Sciences and Arts)</i>	

### **Workshop**

Teaching Novices Programming and Important Applications in a Single Semester - Critical Factors from Zero to Portable GUI Programming in Four Hours .....	xxxvii
<i>Rony G. Fletcher (Vienna University of Economics and Business)</i>	

## Full Papers

Cultivating Software Quality Improvement in the Classroom: An Experience with ChatGPT .....	3
<i>Eman Abdullah AlOmar (Stevens Institute of Technology) and Mohamed Wiem Mkaouer (University of Michigan-Flint)</i>	
How Much Effort Do You Need to Expend on a Technical Interview? A Study of LeetCode Problem Solving Statistics .....	13
<i>Jialin Cui (North Carolina State University), Runqiu Zhang (University of Virginia), Fangtong Zhou, Ruochi Li (North Carolina State University), Yang Song (University of North Carolina Wilmington) and Edward Gehringer (North Carolina State University)</i>	
Leveraging Peer-assessment in Project-based Software Engineering Courses .....	23
<i>Haneen Hijazi (Hashemite University) and Yasser Ali Alshehri (West Virginia University)</i>	
Case Study Based Pedagogical Intervention for Teaching Software Engineering Ethics .....	31
<i>Aditya Johri and Ashish Hingle (George Mason University)</i>	
In the Footsteps of a Professional Software Engineer: Exploring Role-play in Teaching Software Design .....	41
<i>Fatma Meawad (CODE University of Applied Sciences)</i>	
"Think Before You Scrum" - An Experience Report on Daily Scrums to Aid Reflection .....	51
<i>Matthew Minish, Matthias Galster and Fabian Gilson (University of Canterbury)</i>	
Automation of Test Skeletons within Test-Driven Development Projects .....	61
<i>Muhammet Mustafa Olmez and Edward Gehringer (North Carolina State University)</i>	
Leveraging Open Source LLMs for Software Engineering Education and Training .....	71
<i>Juanan Pereira, Juan Miguel López, Xabier Garmendia and Maider Azanza (University of the Basque Country (UPV/EHU)</i>	
Competence-based Assessment of Programming Assignments .....	81
<i>Reinhold Plösch, Iris Groher and Alexander Hofer (Johannes Kepler University Linz)</i>	
Acknowledging Good Java Code with Code Perfumes .....	91
<i>Philipp Straubinger, Florian Obermüller and Gordon Fraser (University of Passau)</i>	
Technology Acquisition Plans to Foster Supply Chain Risk Management Learning Outcomes in Project-Based Software Development Courses .....	101
<i>Bastian Tenbergen (State University of New York at Oswego) and Nancy R. Mead (Carnegie Mellon University)</i>	

## Short Papers

Learning Environment Interoperability in Software Engineering Education .....	113
<i>Dimitri Bigler, Julian Manz, Kevin Lee, David Fischer and Georg Hagel (University of Applied Sciences Kempten)</i>	
Assessing Software Development Competences Constructively Aligned in an Open-Web Format .....	119
<i>Axel Böttcher and Veronika Thurner (Munich University of Applied Sciences)</i>	
Using Generative AI to Create User Stories in the Software Engineering Classroom .....	125
<i>Allan Brockenbrough and Dominic Salinas (Salem State University)</i>	
Adapting Computational Skills for AI Integration .....	131
<i>Hanya Elhashemy (Technical University of Munich), Harold Abelson (Massachusetts Institute of Technology) and Tilman Michaeli (Technical University of Munich)</i>	

A Survey Study on the State of the Art of Programming Exercise Generation using Large Language Models.....	137
<i>Eduard Frankford, Ingo Höhn, Clemens Sauerwein and Ruth Breu (University of Innsbruck)</i>	
Leveraging Lecture Content for Improved Feedback: Explorations with GPT-4 and Retrieval Augmented Generation.....	143
<i>Sven Jacobs and Steffen Jaschke (University of Siegen)</i>	
Workplace Learning Ecology of Software Engineers and Implications for Teaching and Learning.....	149
<i>Aditya Johri, Aayushi Hingle Collier (George Mason University), Brent K. Jesiek (School of Engineering Education, Purdue University), Russel Korte (George Washington University) and Stephen Cory Brozina (Youngstown State University)</i>	
Who is that girl? Drawing a portrait of a female student in Software Engineering.....	155
<i>Yekaterina Kovaleva (LUT University), Praveena Nair Sivasankaran (Taylor's University) and Ari Haaponen (LUT University)</i>	
Automated Programming Exercise Generation in the Era of Large Language Models.....	161
<i>Niklas Meißner, Sandro Speth and Steffen Becker (Institute of Software Engineering, University of Stuttgart)</i>	
Systematic Teaching of UML and Behavioral Diagrams.....	167
<i>Anja Metzner (Technical University of Applied Sciences Augsburg)</i>	
ScrumBoard: A Project Management Tool Purpose-Built for Software Engineering Education.....	173
<i>Matthew Minish, Fabian Gilson and Matthias Galster (University of Canterbury)</i>	
Evaluating Preschoolers' Block Programming Using Complexity and Personality Traits.....	179
<i>Yui Ono (Waseda University), Daisuke Saito (Takachiho University) and Washizaki Hironori (Waseda University)</i>	
Exploration and Practice of University-Enterprise Deep Integration Oriented Talent Training Mode for Software Engineering Specialty.....	185
<i>Shuwei Qiu, Wei Huang, Gengzhong Zheng, Xiaojia Zhou and Mohammad Humayoo (Hanshan Normal University)</i>	
Gameful Introduction to Cryptography for Dyslexic Students.....	191
<i>Argianto Rahartomo, Harpreet Kaur and Mohammad Ghafari (Technische Universität Clausthal)</i>	
Navigating the AI Frontier: A Critical Literature Review on Integrating Artificial Intelligence into Software Engineering Education.....	197
<i>Chandan Kumar Sah, Dr. Lian Xiaoli (School of Computer Science and Engineering, Beihang University), Muhammad Mirajul Islam (College of Technology and Engineering Westcliff University) and Md Kamrul Islam (Université Libre De Bruxelles)</i>	
ChatGPT's Aptitude in Utilizing UML Diagrams for Software Engineering Exercise Generation.....	203
<i>Sandro Speth, Niklas Meißner and Steffen Becker (Institute of Software Engineering, University of Stuttgart)</i>	
Towards Integrating Emerging AI Applications in SE Education.....	209
<i>Michael Vierhauser (University of Innsbruck), Iris Groher (Johannes Kepler University Linz), Tobias Antensteiner (University of Innsbruck) and Clemens Sauerwein (QE Research Group University of Innsbruck)</i>	
Toward AI-facilitated Learning Cycle in Integration Course through Pair Programming with AI Agents.....	215
<i>Zhengyuan Wei, Albert Ting-Leung Lee, Victor Chung-Sing Lee (The University of Hong Kong) and Wing-Kwong Chan (City University of Hong Kong)</i>	

University Students' Perception and Expectations of Generative AI tools for Software Engineering.....	221
<i>Mounika Yabaku and Sofia Ouhbi (Uppsala University)</i>	

A Deep Dive into CATS Evaluator Algorithm: Quantification of The Probability in Serious Game Cloud Security Defense Scenarios.....	227
--	-----

*Tiange Zhao (Siemens AG), Ulrike Lechner (Universität der Bundeswehr München), Maria Pinto-Albuquerque (ISCTE-IUL - Instituto Universitário de Lisboa, Portugal), Didem Ongu and Tiago Gasiba (Siemens AG)*

## Posters

Using Generative Artificial Intelligence Tools in Software Engineering Courses .....	235
<i>Soma Datta (University of Houston Clear Lake)</i>	

You & AI: a research computing hackathon.....	237
---	-----

*James DesLauriers, Katerina Michalickova, John Pinney, Liam Gao and Chris Cooling (Imperial College London)*

MASS, Marburg university auto ASess System.....	239
---	-----

*Steffen Dick and Christoph Bockisch (Philipps Universität Marburg)*

FeedbackPulse: GPT-Enabled Feedback Assistant for Software Engineering Educators .....	241
--	-----

*Yiwen Liao, Yuchao Jiang, Zhangpeng Chen and Basem Suleiman (University of New South Wales)*

Presenting Experiential Educational Machine Learning Labs.....	243
--	-----

*Yang Liu, Xiaofan Que, Dingrong Wang, Samuel A. Malachowsky and Daniel E. Krutz (Rochester Institute of Technology)*

Requirements Engineering Education with Just-in-Time Teaching and Projects.....	245
---	-----

*Ralf Reißing (Coburg University of Applied Sciences and Arts)*

Student Experiences in a Global Software Engineering Course Taught Jointly in Germany, Indonesia and Japan.....	247
---	-----

*Felix Waigner, Andreas Roessler (Nuremberg Institute of Technology), Michael Evan Santoso (Ritsumeikan University), Patricia Brockmann (Nuremberg Institute of Technology), D. Moritz Marutschke (Kyoto University of Advanced Science), Tri Astoto Kurniawan (Brawijaya University) and Victor Kryssanov (Ritsumeikan University)*

## Tool Demos

A Method and Software to Create SCORM Units for Computer Science Courses .....	251
--	-----

*Peter Braun (Technical University of Applied Sciences Würzburg-Schweinfurt)*

Accessible Learning Labs: Accessibility Education Through Experiential Learning.....	253
--	-----

*Yang Liu, Xiaofan Que, Dingrong Wang, Samuel A. Malachowsky and Daniel E. Krutz (Rochester Institute of Technology)*

Automated Assessment Tool for Teaching Web Application Development .....	255
--	-----

*Basem Suleiman (University of New South Wales), Johan Alibasa (Sampoerna University) and Anthony Wang (University of Sydney)*

A Tool to Facilitate Calibrated Peer Reviews in Software Engineering Education .....	257
--	-----

*Bastian Tenbergen (State University of New York at Oswego)*

## Industry Track

Can LLMs Facilitate Onboarding Software Developers? An Ongoing Industrial Case Study.....	261
<i>Maider Azanza, Juanan Pereira, Arantza Irastorza (University of the Basque Country (UPV/EHU) and Aritz Galdos (LKS Next)</i>	
Mastering Software Craftsmanship: An industrial practitioner perspective.....	267
<i>Arastoo Barji (Siemens AG, Digital Industries, Process Automation), Thomas Jachmann (Siemens Healthineers AG, Computed Tomography, Head of Software) and Frances Paulisch (Siemens Healthineers, Head of Software Initiative)</i>	
Experience Report of the AWS+KCL Impact Accelerator for Public Sector Engagement.....	273
<i>Caitlin Bentley, Elena Simperl (King's College London), Mike Bainbridge, Daisy Ogden (Amazon Web Services), Stefanos Leonardos, Gunel Jahangirova, Joanna Walker, Wenjie Yin and Christopher Hampson (King's College London)</i>	
Serious Game for Industrial Cybersecurity: Experiential Learning through Code Review.....	279
<i>Andrei-Cristian Iosif (Siemens AG), Ulrike Lechner (Universität der Bundeswehr München), Maria Pinto-Albuquerque (ISCTE-IUL - Instituto Universitário de Lisboa, Portugal) and Tiago Gasiba (Siemens AG)</i>	

## Journal First Track

Towards understanding students' sensemaking of test case design: a one-page summary .....	287
<i>Niels Doorn (Open Universiteit), Tanja E. J. Vos (Open Universiteit and Universitat Politecnica de Valencia) and Beatriz Marin (Universidad Politecnica de Valencia)</i>	
Summary of: Lessons Learned from a Learning Program for Software Architects.....	289
<i>Frances Paulisch (Siemens Healthineers, Head of Software Initiative), Matthias Backert (Siemens Healthineers AG, Software Initiative, Training), Thomas Blum (Siemens Healthineers AG, Diagnostic Imaging, Magnetic Resonance)</i>	
Extension of a study on Parsons' Problems reveals implications for SEE&T.....	291
<i>Kevin D. Wendt (University of Minnesota)</i>	
Thriving in the era of hybrid work: Raising cybersecurity awareness using serious games in industry trainings.....	293
<i>Tiange Zhao, Tiago Gasiba (Siemens AG), Ulrike Lechner (Universität der Bundeswehr München) and Maria Pinto-Albuquerque (ISCTE-IUL - Instituto Universitário de Lisboa, Portugal)</i>	
Teaching Engineering of AI-Intensive Systems .....	295
<i>Atif Mashkoor (Johannes Kepler University Linz); Wesley K. G. Assunção (North Carolina State University); Alexander Egyed (Johannes Kepler University Linz)</i>	
<b>Author Index .....</b>	<b>297</b>



# Message from General Chair

## CSEE&T 2024

I am delighted to welcome you to the 36<sup>th</sup> International Conference on Software Engineering Education and Training (CSEE&T) in Würzburg, Germany. This year's conference is hosted by the Technical University of Applied Sciences Würzburg-Schweinfurt (THWS) and technically co-sponsored by the IEEE Reliability Society. The conference is held as an in-person event from July 29<sup>th</sup> to August 1<sup>st</sup>, 2024 at THWS.

THWS has multiple campus in the cities of Schweinfurt and Würzburg and is home for more than 9.000 students, with a huge international student population, indicating its international flair although not located in one of Germany's metropolis. The charming city of Würzburg is surrounded by picturesque castles, vineyards, and the Main river. It is located in central Germany, belongs to Lower Franconia – one of Germany's famous wine regions – and the northern part of the Freestate of Bavaria – the largest of Germany's sixteen states.

CSEE&T remains the world leading conference on software engineering education for decades. Software engineering, and particularly, its education is dramatically increasing in their importance due to latest technological advancements, we have seen in the last couple of years. To name just three major challenges, we as a community are facing right now: The rise of AI changes the way we teach, or sometimes how students try to teach themselves, but thereby impacting our courses; we also see an increase in the topics and depth of topics future software engineers need to be educated in due to the new technology; software engineering has spread from being a subject only relevant for computer scientists, we see a spread into other disciplines that we – today – consider software engineers, although they are no traditional computer scientists. An example, therefore, I am teaching software engineering in a robotics degree program at THWS Schweinfurt Campus, the students gain in-depth software engineering skills and are upon graduation tasked to work with and as software engineers, but miss the computer science background due to their rooting in electrical engineering.

Considering all these challenges I am very pleased that these are reflected in this year's conference program carefully selected by the program chairs and the program committee. This shows once again that CSEE&T is a conference, and we as a community, on the forefront of research, seeking out future challenges, and not stuck in the past.

A conference like CSEE&T is not possible without its community. Thus, a vast number of individuals are directly involved in the organization or contributed to CSEE&T 2024. I therefore, like to thank all these people. I thank, IEEE as Publisher and the IEEE Reliability Society as Technical Sponsor, the administration of THWS, and the CSEE&T Steering Committee for their valuable support. I thank all program committee members, the paper authors and those who have entrusted us with their manuscripts, the participants, our keynote speakers for making CSEE&T 2024 a success. I thank Nancy Mead (Carnegie Mellon University) and Yvonne Sedelmeyer (SRH University of Applied Sciences / Coburg University of Applied Sciences) for crafting such a fitting ASEE&T program; Jennifer Brings (Bingen Technical University of Applied Sciences) for bringing together such a vibrant industry

track; Jil Klünder (Leibniz University Hannover) and Rony G. Flatscher (Vienna University of Economics and Business) for taking care of the workshop program; Bastian Tenbergen (State University of New York at Oswego) who cared for the webpage and publicity. Special thanks go to Eric Wong (University of Texas) who dedicatedly took charge of the coordination with IEEE.

Lastly, I need to thank four individuals whose commitment was crucial for CSEE&T 2024 and went far beyond what could be expected. I thank my Program Chairs Andreas Böllin (University of Klagenfurt) and Ivana Bosnić (University of Zagreb) for all their support to the organization of CSEE&T 2024, going beyond creating the scientific program, Meenakshi Manjunath (THWS) and Jeshwitha Jesus Raja (THWS) who as Publication and Local Chairs, were a huge support in bringing this conference to life.

**CSEE&T 2024 General Chair**

Marian Daun

Technical University of Applied Sciences Würzburg-Schweinfurt, Center for Robotics (CERI)

# Message from Program Chairs

## CSEE&T 2024

We are pleased to introduce the 2024 36<sup>th</sup> International Conference on Software Engineering Education and Training (CSEE&T) held in Würzburg, Germany, from July 29<sup>th</sup> to August 1<sup>st</sup>, 2024.

CSEE&T, now in its 36<sup>th</sup> edition, continues to be the leading international conference in software engineering education and training. This year, we have expanded our tracks to encompass a wider range of submissions and insights into the evolving field of software engineering education.

### Submission Overview

This year, we received 107 submissions (in two submission rounds) across various tracks. The breakdown is as follows:

- **Full Paper Track:** We received 85 submissions (35 full and 50 short papers). Of these, 31 were accepted (11 full papers and 20 short papers).
- **Industry Track:** We received 7 submissions, of which 4 were accepted.
- **Journal First Track:** We received 5 submissions, all accepted.
- **Tool/Poster Submissions:** We received 12 submissions and accepted 10.

### Focus of This Year's Conference

The primary focus of this year's conference reflects significant trends and advancements in software engineering education. Based on an analysis of the accepted submissions, the prominent themes include:

- **Large Language Models and AI:** Exploring the applications and impacts of AI, including tools like ChatGPT, on education.
- **Education and Learning Analytics:** Utilizing data-driven approaches to enhance learning outcomes.
- **Testing and Programming Education:** Innovations in teaching programming and testing methodologies.
- **Experiential and Computing Education:** Practical, hands-on approaches to learning.

These themes illustrate our commitment to integrating modern AI technologies and data-driven methods into software engineering education, ensuring we stay at the forefront of educational advancements.

### Industry Track

The industry track plays a vital role in the CSEE&T Conference as it brings together academia and industry. Therefore, we are particularly delighted to feature contributions from industry presenting industry-tested software engineering education and training approaches: promoting professionalism through software craftsmanship, enhancing code review practices with serious games, developing a public sector linked group project module, and using Large Language Models for developer onboarding.

## **ASEE&T**

The Academy for Software Engineering Education & Training (ASEE&T) aims to support educators and curriculum designers in the field of software engineering. This year, the event features distinguished speakers who will discuss current topics, share experiences in curriculum design, and explore strategies for software engineering education. The workshop will begin by identifying the challenges faced by participants in teaching or introducing new ideas into curricula. Following the speakers' insights, attendees will engage in collaborative discussions to develop actionable strategies to address these challenges.

## **Workshop and Panel**

Our conference will feature one workshop and one panel session. Rony G. Flatcher hosts the workshop "Teaching Novices Programming and Important Applications in a Single Semester - Critical Factors from Zero to Portable GUI Programming in Four Hours" and Marian Daun will chair a panel about the "Future of SE Education in the Dawn of AI".

## **Keynote Speakers and Highlights**

We are honoured to have esteemed keynote speakers who will share their insights on crucial topics in the field:

- **Sascha Kegreiß** will discuss „Mastering the Balance Between Software Design Qualities“
- **Nancy Mead** will present „AI and Software Engineering Education: Riding the Wave of Innovation“

We extend our heartfelt thanks to all authors for their submissions, the Program Committee members for their rigorous review work, and all participants for engaging in this conference. We also would like to express our appreciation to the following Session Chairs: Marian Daun, Rony G. Flatcher, Emanuel Grant, Stephan Krusche, Dieter Landes, Meenakshi Manjunath, Nancy Mead, Niklas Meissner, and Bastian Tenbergen for helping out during the conference.

We hope you enjoy this year's program and the rich discussions it will undoubtedly inspire.

## **CSEE&T 2024 Program Chairs**

Andreas Bollin, *University of Klagenfurt, Department of Informatics Didactics*

Ivana Bosnić, *University of Zagreb, Faculty of Electrical Engineering and Computing*

Jennifer Brings, *Zeiss SMT*

# Organizing Committee

## CSEE&T 2024

### **General Chair**

Marian Daun, *Technical University of Applied Sciences Würzburg-Schweinfurt, Germany*

### **Program Chairs**

Andreas Böllin, *Kyoto University of Klagenfurt, Department of Informatics-Didactics, Austria*

Ivana Bosnić, *University of Zagreb, Faculty of Electrical Engineering and Computing, Croatia*

### **Industry Chair**

Jennifer Brings, *Zeiss SMT, Germany*

### **Financial Chair**

Dorit Borrmann, *Technical University of Applied Sciences Würzburg-Schweinfurt, Germany*

### **Local Chair**

Jeshwitha Jesus Raja, *Technical University of Applied Sciences Würzburg-Schweinfurt, Germany*

### **Workshop Chair**

Jil Klünder, *Leibniz Universität Hannover, Germany*

### **Publication Chair**

Meenakshi Manjunath, *Technical University of Applied Sciences Würzburg-Schweinfurt, Germany*

### **ASEE&T Chair**

Nancy Mead, *Carnegie Mellon Software Engineering Institute, United States*

Yvonne Sedelmaier, *SRH Wilhelm Löhe Hochschule in Fürth / Hochschule Coburg, Germany*

### **Publication Chair and Webmaster**

Bastian Tenbergen, *State University of New York at Oswego, United States*

### **IEEE Coordination Chairs**

W. Eric Wong, *University of Texas at Dallas, United States*



# Program Committee

## CSEE&T 2024

Shashikant Albal, *SSN School of Advanced Software Engineering, Indonesia*

Mark Ardis, *Stevens Institute of Technology, United States*

Donald Bagert, *Benedictine College, United States*

Matthew Barr, *University of Glasgow, United Kingdom*

Andreas Bollin, *University of Klagenfurt, Department of Informatics-Didactics, Austria*

Jürgen Börstler, *Blekinge Institute of Technology, Sweden*

Ivana Bosnić, *University of Zagreb, Faculty of Electrical Engineering and Computing, Croatia*

Pierre Bourque, *Ecole de technologie supérieure, Canada*

Patricia Brockmann, *Nuremberg Institute of Technology, Germany*

Qi Cao, *University of Glasgow, United Kingdom*

Jeffrey Carver, *University of Alabama, United States*

Juan Chen, *College of Computer, National University of Defense Technology, China*

Paolo Ciancarini, *University of Bologna, Italy*

Tayana Conte, *UFAM, Brazil*

Igor Čavrak, *University of Zagreb, Faculty of Electrical Engineering and Computing, Croatia*

Leonardo Da Silva Sousa, *Carnegie Mellon University Silicon Valley, United States*

Ronnie de Souza Santos, *University of Calgary, Canada*

Ali Ebnenasir, *Michigan Technological University, United States*

Rob Elliott, *Indiana University Purdue University Indianapolis, United States*

Kevin Gary, *Arizona State University, United States*

John Georgas, *Northern Arizona University, United States*

Emanuel Grant, *University of North Dakota, United States*

Peggy Gregory, *University of Glasgow, United Kingdom*

Georg Hagel, *Hochschule Kempten, Germany*

Noriko Hanakawa, *Hannan University, Japan*

Hiroaki Hashiura, *Nippon Institute of Technology, Japan*

Atsuo Hazeyama, *Department of Information Science, Tokyo Gakugei University, Japan*

Kiyoshi Honda, *Osaka Institute of Technology, Japan*

Liguo Huang, *Southern Methodist University, United States*

Jati H. Husen, *Telkom University, Indonesia*

David Klappholz, *Stevens Institute of Technology, United States*

Verena Klös, *TU Dresden, Germany*

Jil Klünder, *Leibniz Universität Hannover, Germany*

Mate Kovacs, *Ritsumeikan University, Japan*

Stephan Krusche, *Technical University of Munich, Germany*

Dieter Landes, *University of Applied Sciences and Arts, Germany*

Richard Leblanc, *Seattle University, United States*

Timothy Lethbridge, *University of Ottawa, Canada*

Zilu Liang, *Kyoto University of Advanced Science, Japan*

Erina Makihara, *Ritsumeikan University, Japan*

Beatriz Marin, *Universidad Politecnica de Valencia, Spain*

D. Moritz Marutschke, *Faculty of Economics and Business Administration, Kyoto University of Advanced Science, Japan*

Nancy Mead, *Carnegie Mellon University, United States*

Niklas Meißner, *Institute of Software Engineering, University of Stuttgart, Germany*

Miguel Ehécatl Morales Trujillo, *University of Canterbury, New Zealand*

Juergen Mottok, *LaS<sup>3</sup>, OTH Regensburg, Germany*

Masataka Nagura, *Nanzan University, Japan*

Shin Nakajima, *National Institute of Informatics, Japan*

Emily Navarro, *University of California, Irvine, United States*

Shinpei Ogata, *Shinshu University, Japan*

Mark Pault, *The University of Texas at Dallas, United States*

Reinhold Plösch, *Johannes Kepler University Linz, Austria*

Daniel Port, *University of Hawaii, United States*

Damith Rajapakse, *National University of Singapore, China*

Thomas Richter, *Rhein-Waal University of Applied Sciences, Germany*

Benoît Ries, *University of Luxembourg, Luxembourg*

Guoping Rong, *Nanjing University, China*

Hossen Saiedian, *University of Kansas, United States*

Maria-Isabel Sanchez-Segura, *Carlos III University of Spain, Spain*

Yvonne Sedelmaier, *Hochschule Coburg, Germany*

Budi Darma Setiawan, *Brawijaya University, Indonesia*

David Sinclair, *Dublin City University, Ireland*

Derek Somerville, *University of Glasgow, Scotland United Kingdom*

Sandro Speth, *Institute of Software Engineering, University of Stuttgart, Germany*

Tim Storer, *University of Glasgow, Scotland United Kingdom*

Sven Strickroth, *LMU Munich, Germany*

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

Ramin Tavakoli Kolagari, *Nuremberg Institute of Technology, Germany*

Bastian Tenbergen, *State University of New York, United States*

Barrie Thompson, *University of Sunderland, United Kingdom*

Steve Tockey, *Construx Software, United States*

André van der Hoek, *University of California, Irvine, United States*

Charles Wallace, *Michigan Technological University, United States*

Hironori Washizaki, *Waseda University, Japan*

Norihiro Yoshida, *Ritsumeikan University, Japan*



# **Steering Committee**

## CSEE&T 2024

### **Chair**

Daniel Port, *University of Hawaii, United States*

Andreas Bollin, *Alpen-Adria University of Klagenfurt, Austria*

Tony Cowling, *University of Sheffield, United Kingdom*

Marian Daun, *Technical University of Applied Sciences Wurzburg-Schweinfurt, Germany*

Elke Hochmüller, *Carinthia University of Applied Sciences, Austria*

LiGuo Huang, *Southern Methodist University, United States*

David Klappholz, *Stevens Institute of Technology, New Jersey, United States*

Stephan Krusche, *Technical University of Munich, Germany*

Timothy C Lethbridge, *University of Ottawa, Canada*

Nancy Mead, *Carnegie Mellon University, United States*

Bastian Tenbergen, *SUNY Oswego, United States*

Barrie Thompson, *University of Sunderland, United Kingdom*

Hironori Washizaki, *Waseda University, Japan*

Eric Wong, *University of Texas, Dallas, United States*



# **Keynotes**



# Mastering the Balance between Software Design Qualities

(Invited Talk)

Sascha Kegreiß  
[lectures@kegreiss.eu](mailto:lectures@kegreiss.eu)

***Abstract*** — In academic education of software engineers, equipping students with the skills to navigate trade-offs between competing software qualities, such as usability and security, is essential. However, those skills are mostly trained independently and isolated from each other, which may lead to an overrated perception of the one or the other discipline. Training the ability to find an appropriate level of one software quality due to differing conditions and in relation to other relevant qualities is often crucial to a products' success. Collaborative projects can simulate real-world team dynamics, teaching students to negotiate and reconcile different quality priorities. By embedding these elements into the curriculum, academic institutions can produce software engineers who are adept at making informed decisions that enhance user experience with an adequate level of security. This balanced approach in academic training is crucial for developing resilient, user-friendly, and secure systems, preparing students to meet the evolving demands of the software industry.



# AI and Software Engineering Education: Riding the Wave of Innovation

Nancy R. Mead  
*Carnegie Mellon University*  
*IEEE Fellow*  
Pittsburgh, PA, United States  
[nrmcmu@gmail.com](mailto:nrmcmu@gmail.com)

**Abstract**— These days we hear a lot about AI, its rapid development, and the changes that might result. These concerns are not only in the field, but also overflow into the classroom. Generative AI applications are being introduced at all levels of education. Starting in elementary school, students learn to use generative AI applications, causing concern among parents who may be less familiar with such applications than the young students. As students continue their education, they can learn to use AI applications in various situations.

By the time computing students reach the university level, they are very familiar with AI applications. Computing students are concerned about whether they should even bother getting a degree in a computing field, because all the jobs might go away. This also applies to our software engineering students at all levels. Faculty are concerned about whether they will have anything left to teach if they no longer need to teach programming or other software engineering fundamentals the way they do today. Books on AI are springing up all over, as are companies! We will try to separate the hype from the reality of AI and education.

We will also reflect on AI in software engineering bodies of knowledge, such as the newly minted SWEBOK, and how AI can be incorporated into software engineering and related curricula and degree programs. One of our concerns is that AI applications are progressing more quickly than our ability to incorporate them into standard BOKs and curricula. As educators, we have all dealt

with evolution and change in the material that we teach. We can consider how changes have previously been reflected in our bodies of knowledge and curricula and discuss ways in which that prior experience might inform incorporating AI into our software engineering curricula.

**Keywords**—software engineering education, artificial intelligence, AI, bodies of knowledge



**Nancy R. Mead** (SEI Fellow [retired], CMU) is a Fellow of the Software Engineering Institute (SEI), and an Adjunct Professor of Software Engineering at Carnegie Mellon University. She is a Parnas Fellow at Lero, the Irish Software Research Institute. Her research areas are security requirements engineering and software assurance curricula. The Nancy Mead Award for Excellence in Software Engineering Education is named after her. Dr. Mead has more than 200 publications and invited presentations. She is a Fellow of the IEEE, a Distinguished Member of the ACM, was named the 2015 Distinguished Educator by IEEE TCSE and received the 2024 IEEE Women in Services Computing (WISC) Award.



**ASEE&T**



# The Sustainable Competency Oriented Computing Education for Π-Shape Elites and New Forms of Digital Education in the Future

Xiaofei XU, *Member, IEEE*  
Harbin Institute of Technology  
xiaofei@hit.edu.cn

## Abstract

The new generation of information technology has led a new wave of digitalization in the world. The digital transformation of higher education has been promoted by this wave. Facing to the challenges of these new change, universities around the world are actively exploring the new education models and the ways to cultivate Π-shape elites in the digital era. The new forms of digital education, e.g., sustainable competency oriented agile education, AI+ education, service-oriented education, and metaverse education, are presented in this paper.

## Index Terms

computing education, sustainable competence, agile education, digital education, intelligent education, service- oriented education, metaverse education.

## I. INTRODUCTION

The wave of digitalization by new information technologies is changing the manners of human life and production globally. Furthermore, the Generative Artificial Intelligence (GAI) and Large Language Models (LLM) have accelerated the process of digitalization and intelligence of the human society and education. The digital transformation of higher education has been also promoted by this wave. The digital education causes the “digital subversion” in the aspects of ethos, models, teaching, learning, methods, environment, and management of higher education. With deep fusion of digital technologies and modern education, the new forms of digital education, e.g., Internet+ education, Artificial Intelligence (AI) + education, service-oriented education, and metaverse education, are rapidly developing.

Facing to the challenges of these new changes, universities are actively exploring new education models and the ways to cultivate comprehensive elites in the digital era, in order to cultivate students to adapt to the rapid change of technologies and society. In such situation, a question, ‘How to establish a new educational system to ensure the professional competencies of students?’ is presented. This problem deals with the sustainable competencies of students in their professional development. Especially, it is essential for computing education firstly to response this problem.

In this paper, the concept of sustainable competency and the related computing education are presented to adapt to future technological and social changes. The new model of sustainable competency oriented agile education and open education ecosystems are proposed. Then, the new forms of higher education, e.g., AI+ education, service-oriented education, metaverse education, are also discussed.

## II. THE SUSTAINABLE COMPETENCY ORIENTED AGILE EDUCATION FOR THE Π-SHAPE ELITES

In recent years, a problem about promoting the sustainable competency of young people has been considered by the educators in universities. Since 2017, the “Twenty-Experts Forum on Computer Education” in China has been discussing how to cultivate innovative computer talents with adaptability and competence in the face of constantly changing scientific, technological, and social environments. The concept of “Sustainable Competence” has been proposed in this forum, and published in the Blue Book “Computer Education and Sustainable Competence” in 2019. Sustainable competence refers to the adaptability to the future social changes and competition, the creativity for innovation based on mission and technology, and the operation ability to drive social development and technological progress. In the CC2020 by IEEE CS and ACM, a similar concept of “competency” was presented. It is described that competencies are “things” including job-relevant behavior, motivation, and technical knowledge/ skills, that an individual must demonstrate to be effective in a job, role, function, task, or duty.

Considering the interdisciplinary and cross-domain characteristics of modern IT (Information Technology) industry, it is required for universities to cultivate students who not only possess specialized knowledge in a specific field, the I-shaped people, but also have broad knowledge, diverse abilities, and strong comprehensive qualities, the T-shaped comprehensive elites. Besides, beyond the T-shaped people, IT industry further requires Π-shaped comprehensive leading talents, who have both specific knowledge and industrial experience.

In order to cultivate sustainable competency-oriented talents, it is necessary to establish a higher education system capable of fostering students' adaptability and innovation capabilities. "Agile Education" is such a new form of education with high flexibility and dynamic adaptability. Facing to the multi-objectives and individualized demands of people development, and following the learning law, agile education realizes multiple rounds iteration of knowledge learning and ability promotion, by means of concurrent crossing and rapid learning of theory, techniques and practice, and efficient coordinated education by multi-universities and multi-domains educational resource. Agile education is an approach to combine theory, knowledge, ability, and diathesis into a comprehensive education system.

What enterprises seek are the talents who can understand industry development and enterprise's requirements, apply advanced technologies to solve their current and future practical problems. Therefore, it's essential to help students comprehend the computing disciplines, industrial enterprises, to enhance their abilities to integrate interdisciplinary knowledge, to apply theory into practice, and to adopt a problem-solving mindset. Through university-industry collaboration and co-cultivation, the enterprise's demands, technologies, resources, cases, expertise, platforms, and environments can be brought into universities, to enable students to grow in the real industrial scenarios. Fostering students' professional competencies should be integrated throughout the university education processes, enterprise internships and practical training.

In order to establish such an agile education system, an "Open Education Ecosystem" should be established, which deals with the specialized education, agile education, university operation management, and educational resources inside and outside the university, etc. The open education ecosystem will build a student-centric education environment, which can coordinate or integrate the educational resource inside or outside of an university, and even international resource, by means of multi-channels collaboration, for agile education and sustainable competencies. The open education ecosystem is to play the active promotion and constraint roles in engendering, development, and evolution of an education system. The organization, coordination, operation management, governance and compliance are all the important factors of the open education ecosystems.

In order to cultivate II-shaped versatile leadership talents, Harbin Institute of Technology at Weihai Campus has proposed and performed the "II-shaped New Engineering Education Solution" with agile education model.

### III. THE NEW FORMS OF DIGITAL EDUCATION IN THE FUTURE

With support of new information technologies, the new forms of digital higher education appear recently, including AI+ Education, Service-Oriented Education, and Metaverse Education, etc.

The AI+ Education is one of the new forms of higher education. The technologies, platforms, and tools of AI, will empower higher education, and change the educational concepts and models, teaching and learning methods, education organization and environment, management, governance, etc. AI+ education will build a mass individualized, active, precise, student-centric intelligent education system, to deeply improve the quality, efficiency, experience, and effect of education. The GAI and LLM have made bigger influences on computing education than before. The LLM-enabled teaching/learning tools, e.g., the LLM-based self-learning tools, intelligent teaching assistant tools, the LLM-enabled construction tools of teaching contents and resource, smart classrooms, and LLM-enabled Q&A tools, etc., will greatly improve teaching efficiency and learning effectiveness.

Service-Oriented Education establishes a smart education platform and comprehensive networked education service system for mass individualized learners, through integrating university education systems, vocational training systems, social and industrial education related resources, and online education resources. Service-oriented education is a new form of student-centric life-time education system, which provides continuously multi-phases of agile education services for sustainable competencies during the whole professional life of learners.

Metaverse Education is an advanced phase of digital education and e-learning, to facilitate education to provide the student-centric learning space for cross time-space, immersive, interactive, and collaborative learning experiences by means of metaverse technologies, e.g., XR-based real-virtual fusion, AI, block chain, intelligent terminals, etc., in order to enhance learner's perception, memory and ability, to improve educational efficiency and effectiveness. Metaverse Education Services are introduced to provide intelligent services to educational activities and processes, teachers, and students/learners in metaverse environment, to improve learner's experience and learning efficiency, and educational value, quality and effectiveness.

These novel forms of digital education will provide the available model and platform support for computing education, in order to cultivate the II-shaped comprehensive leading talents.

### IV. CONCLUSION

In current development procedure of society and technologies in digital era, the only unchanged status is change. It is required for computing and engineering education to cultivate the T-shaped and the II-shaped comprehensive professionals with sustainable competencies in a changing environment, who can meet the current and future demands of the modern society and IT industry. In order to promote the digital transformation of higher education, it is necessary to update the education concepts, innovate the education models, empower education with AI, reform the education systems, build more digital education resources, enhance the digital abilities of staffs and students, and realize the integration of education ecology. The above new forms of higher education show the valuable models in advancing computing and engineering education, fostering innovative talents, and promoting development of digital education in the future.

# Constructive Alignment in Software Engineering Education

Axel Böttcher, *Senior Member, IEEE*  
HM University of Applied Sciences  
Department of Computer Science and Mathematics  
Lothstraße 64 – 80335 München  
axel.boettcher@hm.edu

## Abstract

The principle of constructive alignment requires that learning objectives, exam objectives and learning methods should be aligned with each other. This talk focuses on the alignment of learning objectives and exams for modern educational settings in software-related courses. As the use of LLM-based tools such as github-copilot or chatGPT has become an integral part of software technology, knowledge about LLM-based tools and their responsible use will also be addressed.

For our courses on software-related topics we as instructors need to formulate learning objectives at different levels of granularity: from module descriptions down to fine-grained objectives for a single lecture or learning nugget. The definition of competence-oriented learning objectives at all levels of granularity is discussed, based on Anderson and Krathwohl's revision of Bloom's six-level taxonomy.

For the assessment process, it is necessary to find appropriate forms of examinations that are constructively aligned with the learning objectives. Assessing programming skills requires forms other than just pen-and-paper exams. Suggestions for hands-on lab exams are discussed as well as tasks that can be used.

## Index Terms

constructive alignment, learning objectives, lab exams, large language models, software engineering.



# Competence-Oriented Elaboration of Software Engineering Education

Dieter Landes, *Member, IEEE*  
Coburg University of Applied Sciences and Arts, Coburg, Germany

## Abstract

Software engineering requires quite a few competences, technical ones along with non-technical ones. This talk presents some examples of how several interlinked modules in software engineering education were modified, refined and enriched over time. Starting out from largely instructive lectures, the modules were enriched with activating elements that aim at making students discover several relevant issues in software engineering by themselves. Thus, they better understand the rationale behind the methods and techniques that are available for resolving these issues. For instance, students work on a group exercise which clarifies that capturing requirements from stakeholders is a communication process. This is followed by several additional assignments in which student teams prepare and conduct a requirements elicitation session with largely realistic external customers. Besides, we offer additional learning elements in an adaptive digital learning environment, in particular short learning videos. This allows students to revisit topics they did not understand properly or to dive deeper into specific issues in a self-directed fashion. This is supported by AI technology which recommends learning elements that best serve the individual needs and characteristics of the individual learner.

## Index Terms

Software engineering education, active learning, inductive learning, self-directed learning, technical competences, non-technical competences.



# **Workshop**



# Teaching Novices Programming and Important Applications in a Single Semester - Critical Factors from Zero to Portable GUI Programming in Four Hours

Rony G. Flatscher

*Department of Information Systems and Operations Management*

*WU (Vienna University of Business Administration)*

Vienna, Austria

[rony.flatscher@wu.ac.at](mailto:rony.flatscher@wu.ac.at)

## I. INTRODUCTORY COURSE “BUSINESS PROGRAMMING”

At the Vienna University of Economics and Business Administration (WU), Bachelor students can learn programming from scratch and become able to create portable GUI programs for *Windows*, *macOS*, and *Linux* in just one semester. This is possible within 60 hours (4 hours per week) of class attendance ("contact hours") and a total teaching load of 200 hours, which corresponds to 8 ECTS (European Credit Transfer System). Several critical success factors make this possible, including the course objectives, structure, organization, applied pedagogical principles, and the programming language used for teaching the introductory course [1].

## II. EVOLUTION

Over the course of more than 35 years, there has been an ongoing teaching project that has been aiming at devising an introductory course for teaching novices successfully the fundamentals of (object-oriented) programming and applying the acquired skills and knowledge to introduce the most essential software engineering application infrastructures that are relevant to businesses and as such to business administration and business information system students (like becoming able to program *MS Office*, to create portable GUIs, portable client/server socket (Internet) applications, portable XML text processing, and portable web scraping programs with *cURL* and the Java *Jsoup* class among other things).

Using participant observation and constantly analyzing the difficulties (and successes) students have been experiencing, comparing the teaching results of experiments (like using different programming languages in various semesters, varying content, identifying the minimum building blocks, apprehension problems to tackle, and the like) over time the goal to reliably teach and apply object-oriented programming in a four-hour introductory course in a single semester has been achieved. A significant contribution to achieving the goal has been exploiting the message paradigm of *ooRexx* when devising, implementing, and constantly improving the packages (like *BSF.CLS* and *UNO.CLS*) and libraries (like *BSF4ooRexx850*, a Java bridge) as that considerably eases the interaction with complex environments for the students.

The introductory course “*Business Programming*” has arrived at a state that allows successfully teaching students without prior programming knowledge the fundamentals of object-oriented programming in a single semester. The

students apply the learned concepts to *Windows* and *MS Office* by the middle of the semester, followed by using the *Java* runtime environment that gets exploited to teach the fundamentals of creating portable GUIs, client/server sockets programs, processing of XML text, and portable programming of *Apache OpenOffice* (and its fork *LibreOffice*).

## III. WORKSHOP GOALS

This workshop aims to transfer all the necessary concepts and knowledge such that the workshop attendees should become able to teach novices object-oriented programming from zero to GUI programming and enable the students to apply their freshly acquired knowledge to interact with essential programming environments and take advantage of them (*Windows OLE*, and *Java* on all platforms), all in a single semester with the “*Business Programming*” course.

## IV. WORKSHOP MATERIALS

All the deployed and developed materials, slides, numerous nutshell examples, and developed packages and libraries are made available as open-source and can be put to work immediately.

## V. STRUCTURE OF THE WORKSHOP

The workshop is planned for four hours and consists of four different parts, each lasting approximately one hour and focusing on different important factors such that workshop attendees become able to understand and assess the concepts of the critical success factors successfully and identify the critical installments of this introductory course:

- Part 1: Goals, cognitive load theory, choosing a programming language and organization of the course,
- Part 2: Programming concepts (*ooRexx*) interacting with *Windows OLE* and the *Java* runtime environment [2]-[4],
- Part 3: Critical installments that determine the success,
- Part 4: Hands-on: installing software, running nutshell examples.

### A. Part 1: Goals, Cognitive Load Theory, Organization

The most important goals (teaching outcomes) that the course needs to achieve are explicated, which, in essence, are the fundamental programming concepts and their application in small real-world examples. Introducing the cognitive load theory allows for reasoning and deciding on programming languages to use for teaching novices, which are not

necessarily popular languages like C++, Java, or Python because being defined as static and strictly typed languages incur a high intrinsic load. The aim to reduce the extraneous load has caused specific organizations and rules for holding the course. In addition, the sequence and content of the slides and the nutshell examples used therein have been constantly assessed and updated to help reduce the extraneous load. The goal is to free the students from the intrinsic and extraneous load as much as possible such that the gained working memory can be put to work for allowing a higher germane load, i.e., to increase the ability to learn faster than would be available otherwise.

This analytical approach allows for realizing that only some programming languages are equally well suited to ease novices' learning of programming as their design and rule sets have different intrinsic load effects. ooRexx allows for teaching fundamental programming concepts faster than comparable popular languages because of its low intrinsic load demands.

#### *B. Part 2: Programming concepts (ooRexx) interacting with Windows OLE and the Java runtime environment*

The dynamic and dynamically typed programming language REXX was created specifically as a language that should be easy to learn, apply, and maintain. Principles like human orientation and English keyword statements that convey their meaning (making REXX programs appear almost like pseudo-code), issuing commands by just denoting them reduces the intrinsic load considerably.

ooRexx, the object-oriented successor to REXX, follows its design principle of "human orientation," keeping the syntax simple and introducing SmallTalk's message paradigm that allows programmers to abstract the usage of objects from the implementation context of those objects by using messages for interaction. The pattern of interacting with objects by messages only allows the students to abstract the interactions with all environments that support this notion.

The Windows version of ooRexx comes with an ooRexx proxy class, OLEObject, that abstracts the interaction with Windows objects (via COM and OLE) to a mere exchange of ooRexx messages. The application of the message paradigm in the form of simple message expressions reduces, therefore, the intrinsic load when interacting with Windows objects. Thus, novices can create programs that interact with MS Excel, MS Word, and any other Windows object via OLE.

In more than 20 years, the author has developed and improved the Java bindings for ooRexx such that it has become possible to interact with Java using an ooRexx proxy class BSF that abstracts the interaction with Java objects to mere exchange of ooRexx messages. This has the effect of reducing the intrinsic load massively when interacting with Java objects.

#### *C. Part 3: Critical installments that determine the success*

Applying constant participant observation over decades and improving the slides and the nutshell examples therein, a programming teacher can easily understand the slides and take them up for immediate use for teaching novices programming. However, in the sequence of the slides, critical installments determine the success of teaching this course. Therefore, learning these installments and what makes them critical for success is important. An example is the installment that introduces the definition and use of classes, which is

overwhelming for novices and, therefore, needs repetition to allow digesting and rethinking.

#### *D. Part 4: Hands-on: Installing Software, Running Nutshell Examples.*

In this part of the workshop, each participant will be able to install the necessary software and the slides on his computer. Depending on the preferences, it can also use portable versions.

Once it has become possible to successfully run the samples and nutshell examples with the installed software, the focus is to pick samples and nutshell examples that will encompass the entire course's content.

## VI. LICENSES

All the software and all the teaching materials are open-source and freely available so that attendees can immediately deploy everything necessary for novice courses.

- [1] R. G. Flatscher and G. Müller, “Business Programming’ – Critical Factors from Zero to Portable GUI Programming in Four Hours ”, Marko Kolakovic, Tin Horvatinovic, Ivan Turcic, Eds. Zagreb: 6th Business and Entrepreneurial Economics 2021, proceedings, pp. 76-82. [https://research.wu.ac.at/files/32933925/2021\\_BusinessProgramming\\_BEE2021\\_accordingToGuidelines.pdf](https://research.wu.ac.at/files/32933925/2021_BusinessProgramming_BEE2021_accordingToGuidelines.pdf) (visited: 2024-06-27).
- [2] R. G. Flatscher, “Proposing ooRexx and BSF4ooRexx for Teaching Programming and Fundamental Programming Concepts”, X. Guo, ed. Plano, Texas, 36<sup>th</sup> Information Systems Education Conference (ISECON 2023), proceedings, pp. 89-102. [https://research.wu.ac.at/files/41301564/ISECON23\\_Flatscher\\_Proposing\\_oOrex Article.pdf](https://research.wu.ac.at/files/41301564/ISECON23_Flatscher_Proposing_oOrex Article.pdf) (visited 2024-06-27).
- [3] R. G. Flatscher, “Business Programming Slides, Part 1”, (first half of semester/two months, from zero programming to Windows, MS Office or OpenOffice/LibreOffice programming). (visited: 2024-06-27) <https://wi.wu.ac.at/rfg/wu/lehre/autowin/material/foils/>, unpublished.
- [4] R. G. Flatscher, “Business Programming Slides, Part 2”, (second half of semester/two months, BSF4ooRexx camouflaging Java as ooRexx allows for exploiting all Java class libraries), (visited: 2024-06-27) <https://wi.wu.ac.at/rfg/wu/lehre/autojava/material/foils/>, unpublished.
- [5] T. Winkler and R. G. Flatscher, “Cognitive Load in Programming Education: Easing the Burden on Beginners with REXX.” Dubrovnik: 34<sup>th</sup> Central European Conference on Information and Intelligent Systems, 2023, proceedings, pp. 171-178. [https://research.wu.ac.at/files/46150789/CECIIS\\_CLT\\_REXX.pdf](https://research.wu.ac.at/files/46150789/CECIIS_CLT_REXX.pdf) (visited: 2024-06-27).
- [6] R. G. Flatscher and T. Winkler, „Concepts that Allow Learning the Programming Language Rexx Much Faster than Other Languages“, Karolj Skala and Vedran Mornar, Eds. Opatija: 47<sup>th</sup> ICT and Electronics Convention, proceedings, pp. 1494-1498. [https://research.wu.ac.at/files/64505014/mipro24\\_9192\\_Flatscher\\_Winkler\\_LearningProgrammingFast\\_final-5.pdf](https://research.wu.ac.at/files/64505014/mipro24_9192_Flatscher_Winkler_LearningProgrammingFast_final-5.pdf) (visited: 2024-06-27).
- [7] R. G. Flatscher and T. Winkler, „Employing the Message Paradigm to Ease Learning Object-oriented Concepts and Programming“, Karolj Skala and Vedran Mornar, Eds. Opatija: 47<sup>th</sup> ICT and Electronics Convention, proceedings, pp. 1488-1493. [https://research.wu.ac.at/files/64505159/mipro24\\_9194\\_Flatscher\\_Winkler\\_EmployingMessageParadigm\\_final-4.pdf](https://research.wu.ac.at/files/64505159/mipro24_9194_Flatscher_Winkler_EmployingMessageParadigm_final-4.pdf) (visited: 2024-06-27).

# **Full Papers**



# Cultivating Software Quality Improvement in the Classroom: An Experience with ChatGPT

Eman Abdullah AlOmar\*, Mohamed Wiem Mkaouer†

\*Software Engineering Department, Stevens Institute of Technology, Hoboken, NJ, USA

†Computer Science Department, University of Michigan-Flint, Flint, MI, USA

ealomar@stevens.edu, mmkaouer@umich.edu

**Abstract**—Large Language Models (LLMs), like ChatGPT, have gained widespread popularity and usage in various software engineering tasks, including programming, testing, code review, and program comprehension. However, their effectiveness in improving software quality in the classroom remains uncertain. In this paper, our aim is to shed light on our experience in teaching the use of Programming Mistake Detector (PMD) to cultivate a bugfix culture and leverage LLMs to improve software quality in educational settings. This paper discusses the results of an experiment involving 102 submissions that carried out a code review activity of 1,230 rules. Our quantitative and qualitative analysis reveals that a set of PMD quality issues influences the acceptance or rejection of the issues, and design-related categories that take longer to resolve. Although students acknowledge the potential of using ChatGPT during code review, some skepticism persists. We envision our findings to enable educators to support students with code review strategies to raise students' awareness about LLMs and promote software quality in education.

**Index Terms**—large language models, education, bugfix, code quality

## I. INTRODUCTION

Linting is one of the code inspection practices in which developers leverage static analysis to identify bad coding patterns, also known as issues. These issues have been known to hinder code quality, making it harder to understand and more prone to errors. Since their inception, linters have been introduced early to students, as part of conceptualizing the avoidance of bad programming practices [17]. Yet, linters output is in the form of warnings with no recommended fix. Due to the non-actionable nature of these warnings [25], and the lack of their comprehension [24], many developers end up considering them as false positives [12].

However, the rise of Large Language Models (LLMs), such as the Generative Pre-trained Transformer, which is the core model behind ChatGPT, has gained popularity due to its generative ability to create responses and design solutions for various input problems. Therefore, the use of ChatGPT in education has become an area of debate about its opportunities and threats to student learning [18]. Given the wide applications of LLMs in source code, including code quality [22], programming [4], bug management [11], and program comprehension [15], it presents an opportunity to bridge the gap between developers and the proper adoption of static analysis. In this context, little is known about ChatGPT's ability to help students effectively comprehend and address static analysis outcomes, as part of learning code inspection.

In this paper, we reflect on the experience of integrating ChatGPT in the code linting process, to support students with their task of debugging and improving the quality of existing systems. Specifically, students are instructed to leverage PMD<sup>1</sup>, a state-of-the-art static analysis plugin, to inspect an open source system (code that is not theirs) and identify potential issues belonging to several categories ('Documentation', 'Design', 'Security', etc.). Then, for each identified issue, the student needs to discuss whether it is worth fixing, and so, a corrective action, in the form of a code change, is provided. The student is allowed to use ChatGPT as an assistant when reasoning over the decision of whether to fix an issue and how it should be addressed. In other terms, the language model explainable ability is solicited, along with code generation and debugging. These experiments aim to achieve pedagogical goals related to developing students' code debugging capabilities, which are generally known to be learned in industry [14]. It also familiarizes students with bad coding practices and how to refactor them. Furthermore, it trains students not only to use ChatGPT, but also to review its recommendations, and reason over their validity, with respect to analyzed code, before making a decision of whether and how it should be fixed. This last goal is of particular interest to us, as it gradually elevates students' inherent assumption of ChatGPT's ability to hold the ground truth. In fact, students experience how the language model's code suggestions may not address the issues reported by the PMD tool.

This paper contributes to promoting the wider acceptance of static analysis warnings and leveraging LLMs to improve software quality in educational settings by (i) designing a practical assignment for improving the quality of software systems, and (ii) reporting the experience of using the PMD tool and ChatGPT in software quality assurance course that has been taken by 102 undergraduate and graduate students. As part of the contributions of this paper, we provide the assignment description and the tool documentation for educators to adopt and extend<sup>2</sup>.

The remainder of this paper is organized as follows. Section II outlines our experimental setup. Section III discusses our findings, while the reflection is discussed in Section IV. Section V reviews the existing studies. Section VI captures

<sup>1</sup><https://pmd.sourceforge.io>

<sup>2</sup><https://refactorings.github.io/education/>

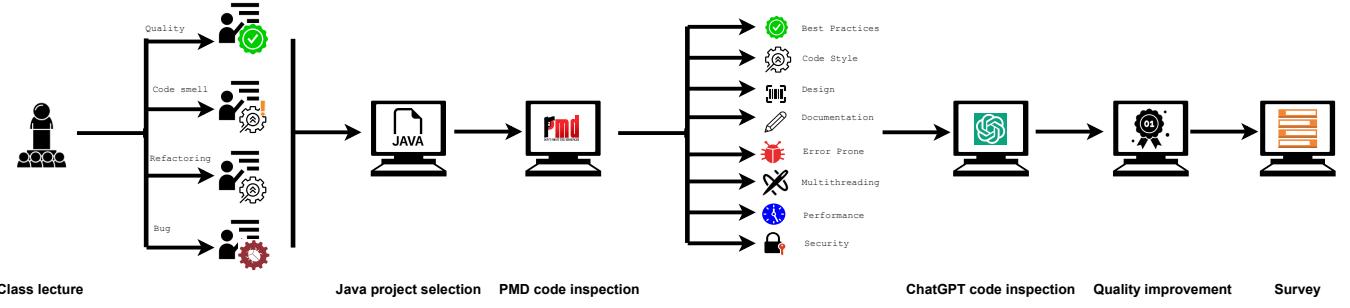


Figure 1: Key phases of our study.

any threats to the validity of our work, before concluding with Section VII.

## II. STUDY DESIGN

### A. Goal & Research Questions

We formulate the main objective of our study based on the *Goal Question Metric* template [20], as follows:

**Analyze the use of PMD for the purpose of cultivating the culture of bug fix and demonstrating a practical case study of leveraging Large Language Models (LLMs) with respect to support educating software quality from the point view of educators in the context of undergraduate and graduate students in SE/CS who analyze Java-based software projects.**

According to our goal, our aim is to answer the following research questions:

- **RQ<sub>1</sub>. What PMD-related problems are typically selected by students?**

Motivation: This RQ explores what type of issues students are addressing using the help of the model.

Measurement: We report the types of issues that are being addressed, clustered by PMD ruleset categories.

- **RQ<sub>2</sub>. What type of issues typically takes longer to be fixed?**

Motivation: This RQ investigates which PMD ruleset takes a longer time to be fixed by students despite the use of ChatGPT. The finding raises educators' awareness of types of issues that are difficult for students to understand and address.

Measurement: We examine the resolution time taken by students to fix each ruleset category.

- **RQ<sub>3</sub>. To what extent was ChatGPT successful in addressing the students' debugging needs?**

Motivation: This RQ explores the ability of the language model to correctly explain the rationale of errors, and to provide potential code fixes when requested. The findings will inform educators about how ChatGPT can support students with code improvement.

Measurement: We examine students' feedback to extract the necessary ratios of the language model's success in addressing their queries.

As part of this paper's contributions, we provide the assignment description, dataset, and tool documentation for educators to adopt and extend<sup>3</sup>.

### B. Course Overview

Software quality assurance is an undergraduate and graduate course consisting of two weekly lectures, one hour and 15 minutes each. The course explores the foundations of software quality and maintenance and introduces challenges linked to various aspects of software evolution, along with support tools to approach them. The course also covers various concepts related to software analysis and testing, along with practical tools widely used as industry standards. Students were also given several hands-on assignments on software quality metrics, code refactoring, bug reporting, unit and mutation testing, and technical debt management. The course deliverables consisted of five individual homework assignments, a research paper reading and presentation, and a long-term group project.

### C. PMD

The Programming Mistake Detector (PMD) is an open-source static source code analyzer. It inputs source code from up to 16 languages and reports common programming issues, such as unsafe threading, god classes, and naming convention violations. These issues are clustered into 8 categories, namely: 'Best Practices', 'Code Style', 'Design', 'Documentation', 'Error Prone', 'Multithreading', 'Performance', and 'Security'. Each issue is identified using a detection rule. The tool has become popular, as it can be integrated with modern CI/CD servers.

### D. Teaching Context and Participants

The study involves one assignment in the software quality assurance course. The course was taught at Stevens Institute of Technology and Rochester Institute of Technology. Before conducting the assignment, students have already learned about several code, and design quality concerns: (1) code quality (teaching quality concepts and how to measure software

<sup>3</sup><https://refactorings.github.io/education/>

```

1. Original Java method containing PMD violations
2. ChatGPT shared conversation
3. Prompt truncated for paper presentation
4. Modified Java method based on the ChatGPT response

```

**Original Java method containing PMD violations:**

```

1. public void paintCalendar201d(Graphics g) {
    Multiple markers at this line
    ShortVariable: Avoid using short variable like 'g'
    CognitiveComplexity: Method 'paintCalendar201d(Graphics)' has cognitive complexity of 50
    NestedControlFlow: Method 'paintCalendar201d(Graphics)' has a NCSS line count of 123
    MethodArgumentCouldBeFinal: Parameter 'g' is not assigned and could be declared final
    CyclomaticComplexity: Method 'paintCalendar201d(Graphics)' has cyclomatic complexity of 12
    ExcessiveMethodLength: Avoid long methods
    NPathComplexity: Method 'paintCalendar201d(Graphics)' has an Npath complexity of 1140
    ...
    208 lines of code truncated for paper presentation
    ...
    24. g.setColor(arrayColor[0]);
}

```

**ChatGPT shared conversation:**

How do I reduce the 'ExcessiveMethodLength' in this method body?

MethodSignature: paintCalendar201d(Graphics)

int headerY = 45;  
int sizeX = 45;  
int sizeY = 45;  
int sizeZ = 45;  
int sizeW = 45;  
int sizeH = 45;

To address the 'excessiveMethodLength' issue in the provided code, you can refactor 'paintCalendar201d' method by breaking it down into smaller, more focused methods. This will improve code readability and maintainability. Here's a refactored version:

```

1. public void paintCalendar201d(Graphics g) {
    int sizeX = 45;
    int sizeY = 45;
    int sizeZ = 45;
    int sizeW = 45;
    int sizeH = 45;
}

```

**Prompt truncated for paper presentation:**

In this refactored version, the code is organized into smaller methods, each responsible for a specific aspect of the drawing logic. This approach improves code readability and makes it easier to understand and maintain.

**Modified Java method based on the ChatGPT response:**

```

1. public void paintCalendar201d(Graphics g) {
    Multiple markers at this line
    ShortVariable: Avoid variables with short names like 'g'
    MethodArgumentCouldBeFinal: Parameter 'g' is not assigned and could be declared final
    CyclomaticComplexity: Method 'paintCalendar201d(Graphics)' has cyclomatic complexity of 12
    NestedControlFlow: Method 'paintCalendar201d(Graphics)' has a NCSS line count of 123
    MethodArgumentCouldBeFinal: Parameter 'g' is not assigned and could be declared final
    CyclomaticComplexity: Method 'paintCalendar201d(Graphics)' has cyclomatic complexity of 12
    ExcessiveMethodLength: Avoid long methods
    NPathComplexity: Method 'paintCalendar201d(Graphics)' has an Npath complexity of 1140
    ...
    208 lines of code truncated for paper presentation
    ...
    24. g.setColor(Color.black);
}

```

Figure 2: ChatGPT in action showing the resolution of ExcessiveMethodLength PMD issue.

quality), (2) code smells (teaching bad programming practices that violate design principles), (3) code refactoring (teaching refactoring recipes that help improving software quality), and (4) bug management (teaching software bugs and how to locate and fix them). The assignment constituted 7.5% of the final grade. It was due 14 days after the four corresponding sessions.

In total, 102 students completed the assignment. The programming experience of the respondents ranged from 1 to more than 10 years, their Java experience ranged from 1 to 10 years, and their industrial/coop experience ranged from 0 to more than 10 years.

#### E. Assignment Content and Format

**Assignment Design.** Initially, students are asked to analyze one version of Java software approved by the instructor to ensure its eligibility based on popularity, in addition to ensuring that it compiles correctly since PMD requires it. The rationale behind giving students the choice of project is to let them choose one that they are comfortable with and that fits their interests. For students who do not want to search for a project, they have been given a list that the instructor has already curated (see Table I). We selected these projects because we already know that they contain a variety of software defects. The students are then asked to set up and run PMD to analyze the chosen project production code. Students are also given the choice of running either the stand-alone version of the tool or its plugin associated with a popular IDE (Eclipse), as we want the students to be familiar with the coding environment and reduce the setup overhead. Upon running PMD, students must choose a minimum of 10 warnings, and at least one from each category, if applicable. We enforce the diversification of warnings to ensure a wider exposure to potential issues, varying from design to multithreading, and documentation. It also increases students' learning curve as they cannot reuse their fix to address multiple instances of the same warning. Yet, we allow students to choose the instances they want to address. It implicitly makes students read many warnings, from all categories, which increases the likelihood of incidental learning. Furthermore, letting students choose the code fragments to review increases their confidence in their decision regarding the given warning. Finally, the students use ChatGPT to fix PMD issues supported by an online questionnaire. For our survey design, we followed the guidelines proposed by Kitchenham and Pfleeger [13]. The

survey is divided into 2 parts. The first part of the survey includes demographic questions about participants. In the second part, we ask about their experience using ChatGPT to fix PMD issues and the perceived usefulness of ChatGPT. As suggested by Kitchenham and Pfleeger [13], we constructed the survey using 6 open-ended questions and 16 multiple-choice questions with an optional “Other” category, allowing respondents to share thoughts not mentioned in the list.

**Pilot Study.** We conducted a pilot study with two students to improve the instrumentation of the experiment and to ensure that the experiment’s instructions were clear. Following the pilot study, we interactively refined the protocol and the assignment questions. The participants in the pilot study are undergraduate students in Software Engineering. Among others, we learned that prompt engineering is crucial when using ChatGPT, and we decided to refine the survey questions to better explore the topic. Therefore, we reformulated the assignment and excluded the data from the pilot study from our analysis.

In a nutshell, the students followed these steps:

- 1) Install the PMD.
- 2) Run PMD on a project of students’ choice and select 10 issues of different types.
- 3) Use ChatGPT to analyze the issues and decide on whether to fix them and what is the appropriate code change action.
- 4) Report the findings for each issue: (1) the source code, (2) the type of issue, (3) how long it took to check it / fix it, and (4) the code snippet.
- 5) Add to the report a concise comment on the experience with ChatGPT (optional).

The evaluation of the artifacts of the submissions was based on 1) the assessment of the students’ ability to understand the type of issues (concept understanding); and 2) assessment of whether students have provided acceptable fixes or proper justification in both cases or accepted or rejected PMD’s recommendation (program analysis and evolution). Students’ perception of the code was excluded from the evaluation process, as it can bias the experiment, as students would be filling out the survey arbitrarily, under the pressure of being graded. Additionally, feedback was anonymous and was not mandatory to increase the magnitude of PMD and ChatGPT usage experience. Although feedback was optional, all students completed it. Figure 1 shows an overview of the setup and

Table I: The list of open-source projects used in the assignment.

Project	# commits	# contributors	Domain
Ant	14,887	64	Java builder
GanttProject	4,361	38	Project management
Hutool	4,074	191	Code design
JCommander	1,009	64	Command line parsing
JFreeChart	4,218	24	Data visualization
JHotDraw	804	3	Data visualization
Log4J	12,211	137	Logging
Nutch	3,293	46	Web crawler
Riho	4,119	80	Script builder
RxJava	6,004	289	Java VM
Xerces	6,463	5	XML parser

execution of our experiment.

#### F. ChatGPT Usage

Just like PMD, students were provided with the necessary background training to use ChatGPT. Since we are interested in students interaction with the language model, students were clearly instructed to fully disclose the usage of the tool through a given survey link. Figure 2 shows one of the PMD issues when analyzing the GanttProject<sup>4</sup>. This issue is called ExcessiveMethodLength, identified in the paintCalendar201d(Graphics g) method in the GanttGraphicArea.java file. PMD highlights this issue with ‘Urgent’ violation due to its CognitiveComplexity of 50, CyclomaticComplexity of 39, and NPathComplexity of 1140, all of which surpass the default acceptance threshold. ChatGPT indicates that refactoring is required to split the lengthy method into smaller and more manageable ones. It also provides a suggested code fix. The student has the option to implement the suggested fix if they agree with it. As shown in the figure, when the proposed fix was adopted, it removed CognitiveComplexity and NPathComplexity issues, and reduced CyclomaticComplexity to 12.

### III. RESULTS

#### A. What PMD-related problems are typically selected by students?

Upon analyzing students’ assignment solutions, we cluster the issues according to the PMD ruleset categories listed in the PMD official documentation<sup>5</sup>, namely, ‘Best Practices’, ‘Code Style’, ‘Design’, ‘Documentation’, ‘Error Prone’, ‘Multithreading’, ‘Performance’, and ‘Security’. These categories were captured at different levels of granularity (*e.g.*, package, class, method, and attributes).

Looking at the PMD rules, Figure 3 depicts the percentages of PMD rules, clustered by category. As can be seen, the most common PMD ruleset category concerns ‘Code Style’, representing 32.3% of the issues. This observation is in line with the findings of previous studies describing that most code reviewers look for style conformance when evaluating

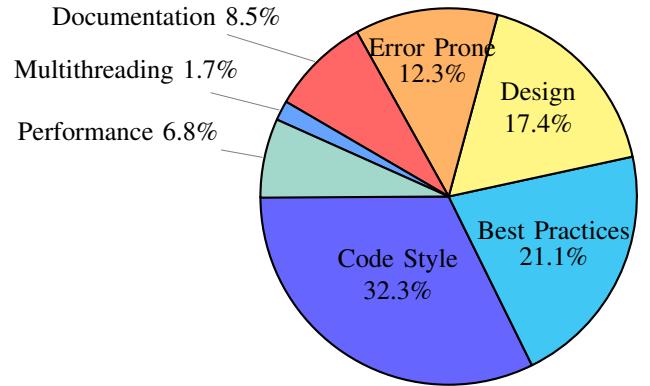


Figure 3: What PMD-related category have you chosen?

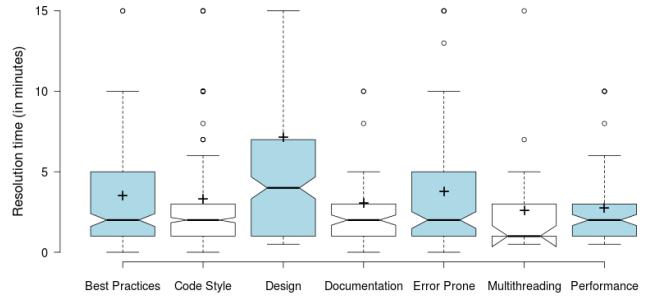


Figure 4: Boxplots of time taken to fix issues, clustered by PMD ruleset categories.

the quality of code [19]. The next most common categories are ‘Best Practices’, ‘Design’, and ‘Error Prone’, representing 21.1%, 17.4%, and 12.3% of the issues, respectively. This might indicate that students have different perspectives on whether developers follow the best practices, improve the architecture design of code, or make code less susceptible to errors. The categories ‘Documentation’, ‘Performance’, and ‘Multithreading’ had the least number of issues, with a ratio of 8.5%, 6.8%, and 1.7%, respectively.

**Summary for RQ<sub>1</sub>:** Among the 1,230 analyzed issues, the most common PMD ruleset category concerns ‘Code Style’, representing 32.3% of the issues.

#### B. What type of issues typically takes longer to be fixed?

Figure 4 shows that issues belonging to the ‘Design’, ‘Error Prone’, and ‘Best Practices’ categories ( $\mu = 7.19$ ,  $\mu = 3.81$ , and  $\mu = 3.51$ , respectively) tend to be more time-consuming for students to address. In particular, ‘Design’ issues have a significant spike compared to all other types. When taking a deeper look at these issues, we noticed this pattern: submissions where students could not receive support with addressing some type of ‘Design’ issues, namely GodClass issues. God Classes are known to be abnormally large, with a concentration of various responsibilities, by controlling and

<sup>4</sup><https://github.com/bardsoftware/ganttproject>

<sup>5</sup><https://pmd.sourceforge.io>

monitoring many other classes. Their existence hinders program maintenance and comprehension [3]. We speculate that design issues take significantly longer to be resolved due to substantial modifications that must be made for God classes. In fact, fixing a God class requires its decomposition into multiple classes, which requires separating multiple tangled concerns while maintaining their coupling to other classes. The needed fix translates into a series of refactorings that go beyond changing one or a few instructions, like when fixing other types of issues. When analyzing students' prompts for fixing for God classes, we noticed that ChatGPT was unable to propose a fix, mostly because the input windows of the model cannot receive the needed set of instructions, which represent the God class, and all its dependencies. Providing a snapshot of the God class code would not be sufficient for the model to carry out a class extraction. The language model was only able to provide examples of extract class refactorings, for relatively smaller-scale classes. For instance, Figure 5 exhibits ChatGPT's response to a student query asking to address a God Class issue, for a given input class. In general, ChatGPT has input limitations when grasping the various contexts in large code segments. Due to this limitation, there have been instances where ChatGPT makes suggestions based on misunderstandings or false assumptions about the code. Through this experiment, students can experience how ChatGPT performance varies based on the length of the input. Increasing the input size can eventually lead to catastrophic forgetting [10].

 I apologize, but I cannot modify the code as I don't have access to the entire codebase or know what the code is supposed to do. However, I can provide an example of how to address the PMD CommentSize issue based on the code snippet you provided:

Figure 5: ChatGPT response to a student query to refactor a God Class.

On the other hand, ChatGPT provided more positive answers to student questions related to addressing issues when they are located in a continuous set of instructions (single location), such as CyclomaticComplexity. ChatGPT was able to propose the extract method refactoring to break down methods with a high cyclomatic complexity score.

**Summary for RQ2:** ‘Design’ PMD ruleset category takes longer to resolve. Unlike other PMD category, this requires going beyond one or few instructions, and ChatGPT has a limited understanding of the broader context and entirety of the codebase.

#### C. To what extent was ChatGPT successful in addressing the students' debugging needs?

Figure 6 shows students' experience with utilizing ChatGPT in terms of four aspects: (1) the ChatGPT's capability to address issues, (2) the potential for ChatGPT to introduce regression in the code, (3) the frequency of interaction with ChatGPT for bug-fixing purposes, and (4) the possibility for

ChatGPT's to introduce additional violations through PMD. When we asked students, “Was ChatGPT able to fix the buggy code?”, 85.9% of the students indicated that ChatGPT can help with addressing the issues, while 14.1% of the students responded negatively. Overall, students are satisfied with ChatGPT's support. In terms of causing regression in the code, 79.6% of the students revealed that there is no regression, while 20.4% of students experienced it. Regarding the frequency of the usage and interaction with ChatGPT before finding (or not) a fix to the buggy code, 40% used it once, 21.3% used it twice. The remaining students (38.7%) used ChatGPT between three and more than five times. As for the potential to introduce additional violations through PMD, the majority of students (86.4%) mentioned that ChatGPT does not contain other errors detected by PMD, but some indicated otherwise.

The success of using any language model heavily depends on students' ability to properly prompt it. Therefore, one of these work's outcomes is sensitizing students to the power of prompt engineering in the context of debugging. As shown in Figure 7, when analyzing the students initial prompts, 27.7% of students revealed that they only copy-paste the buggy code fragments, thinking that ChatGPT can identify the intention behind prompting code. The majority of students (68.6%) communicated that they copied the buggy code fragment along with the error description to provide context for the pasted code. 20.6% of students mentioned that they copy-paste the buggy code fragment and added textual description of how to fix it. A few students reported that they show pairs of buggy and fixed code fragments and asked ChatGPT to do similar for the given buggy code. Besides these activities, 2.9% of the students mentioned in the “other” option: “asking ChatGPT that the PMD violation still remained when it could not fix the code”, “asking ChatGPT how to fix the error in general before copy pasting the offending code”, “prompted ChatGPT with the actual line that was causing the error and asked it if there was something specific in the code segment that could be optimized”, “asked ChatGPT to provide an example of how a similar situation could be fixed”, and “ChatGPT says this is a false positive and is descriptive of what the variable is representing and I agree with it”. Figure 7 shows how the students' prompts are not uniform, where the majority of students argue over how to extract the necessary action from the models, while others overestimate the capabilities of the model, as outlined in previous studies that have proven that ChatGPT is susceptible to hallucination when it comes to coding semantic structures [15], [22]. So, experiencing the potentially inappropriate results of the model would raise the students' awareness of its limitations. In addition, it helps students refine their prompts, as shown in Figure 6 where the majority of students have used more than one prompt per issue (60%).

Furthermore, 68.6% of the students' prompts were zero-shot, *i.e.*, students rely on the generative ability of the model to either understand an issue, or to propose its corresponding code fix. Zero-shot learning challenges the model to make a

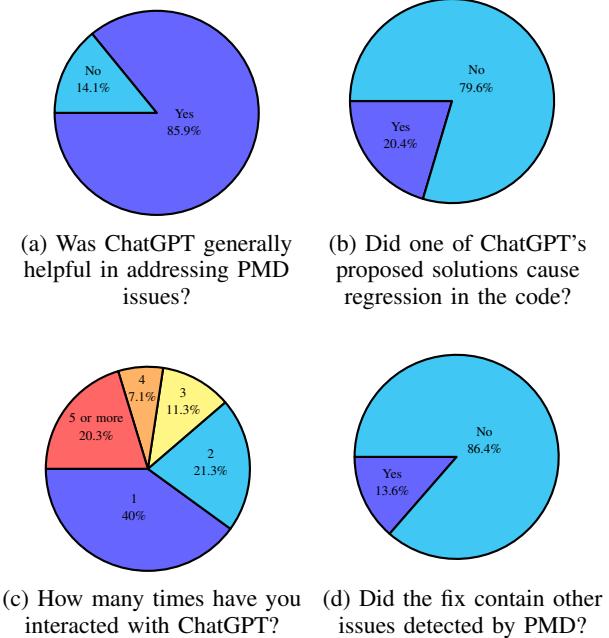


Figure 6: Student's attitude toward utilizing ChatGPT for bug resolution.

decision over presumably unseen data [23]. The model relies on approximating the input with previously trained code. For instance, as shown in Figure 8, the student asks the model to reduce the cyclomatic complexity of an input source code. The same prompt can be augmented by adding a label to the unseen data, *i.e.*, one-shot [8]. For example, in Figure 8, mentioning how the reduction can be performed can guide the model towards the decision to take (method extraction). Other students opted for a few-shot learning by entering pairs of buggy and fixed code fragments and asked ChatGPT to propose similar fixes for an alternative buggy code. Few-shot learning is a response when dealing with complex tasks, to steer the model towards better decision making, by allowing in-context learning from provided examples [21]. In Figure 8, the input shows examples of code changes that address the complexity of a given method.

Given the variety of prompts used, our activity has triggered the students' reasoning about how to query the model to avoid its own inherent limitations. This also helps students be critical of ChatGPT responses, and not treat them as ground truth. For instance, 20.4% of students experienced regression in their code base after using the code generated by ChatGPT. Reported examples ranged from the introduction of compiler errors to failing unit tests. Such problems can be due to existing errors in the ChatGPT generated code or due to the wrong integration of the fix by students into the code base. It can also be eventually caused by an incorrect student query. Although it would be interesting to dive into this analysis more in-depth, our goal was to sensibilize students to how the process of querying language models can be error-

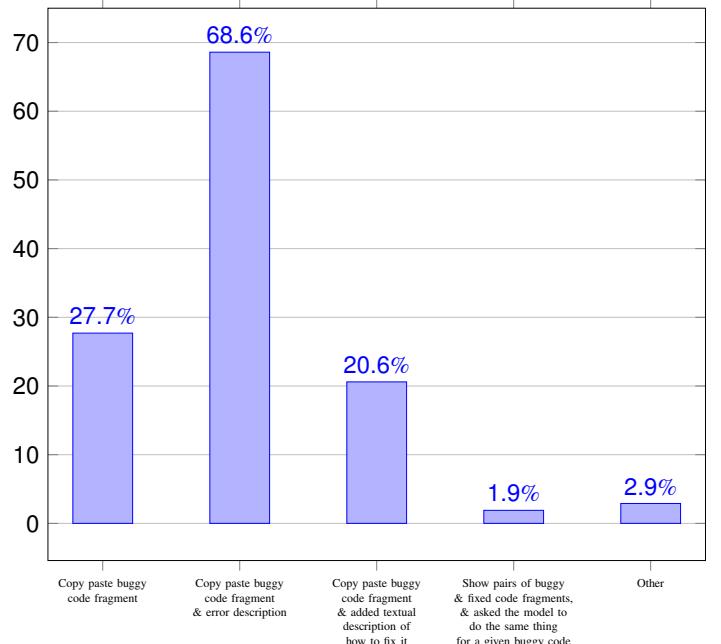


Figure 7: How did you use ChatGPT to fix buggy code?

prone, and thus it is critical to verify its outcomes before any adoption. We also aimed to sensitize students to the existence of multiple querying techniques, which are known in data science curricula, without providing any prompt engineering training.

In Table II, we report the main thoughts, comments, and suggestions about the overall impression of the usefulness, usability, functionality, and recommendation of the ChatGPT, in accordance with the conducted labeling. The table also presents samples of the students' comments to illustrate their impressions of each theme.

**Usefulness.** Generally, the respondents found the ChatGPT to be useful in regard to five main aspects: automation, quality, scope, awareness, and experience. A group of students commented on ChatGPT's ability to automatically fix bugs, its affordability, capacity for automating tedious debugging and bug fixing processes, and ability to deliver precise and thorough responses. Nearly 85.9% of the students commented that ChatGPT is useful as it gives explanations as to what the problem was and the suggestions to fix it. Further, 23.22% of students commented that it is a nice application and provides a good learning experience especially for those who are beginners in software quality and debugging. Some students (20.64%) revealed that ChatGPT was fast in terms of analyzing issues located in contiguous instructions, but not for identifying larger issues such as GodClass and DataClass. 4.5% of students communicated that detecting the issues assists in increasing its *readability*, which helps improve overall code quality.

**Usability.** Based on the feedback the students provided, the key usability areas were documentation, ease of use, explainability and user interface. 28.38% of the students pointed out

### **Zero-shot learning:**

You are a Java expert. How do I fix this section of code that has a cyclomatic complexity of 21:  
“*Original code*”

### **One-shot learning:**

You are a Java expert. Can you fix this section of code that has a cyclomatic complexity of 21 by breaking down the code into smaller methods:  
“*Original code*”

### **Few-shot learning:**

You are a Java expert. I provide you with a pair of fixed and buggy code fragments, and I request that you apply similar fixes to a newly given buggy code.

An example of buggy code that has a cyclomatic complexity of 21:

“*Original buggy code*”

An example of fixed:

“*Original fixed code*”

Here is the newly buggy code, please propose similar fixes for this buggy code:

“*Newly buggy code*”

Figure 8: Different types of learning settings used by students when prompting ChatGPT in the context of debugging.

that the tool is easy to use, has a user-friendly interface, and explains how it generated each bugfix and how to avoid that issue in the future. Other comments also stated that using these LLMs requires to carefully adjust the model’s output. It lacks originality and might give biased output.

**Functionality.** According to the students’ feedback about the tool’s functional features, 16.77% of the students’ comments appreciate ChatGPT’s correction, detection, and debugging functionality and are satisfied with various aspects of the tool’s operation, and how this feature helps to understand bad programming practices in real-world scenarios better. Additionally, the students commented on their ability to practice a variety of strategies to eliminate issues.

**Recommendation.** From the students’ feedback, we have also extracted suggestions to improve the LLMs. 3.8% of the students’ comments show a couple of suggested changes as a recommendation to be made to the tool’s operation.

We extracted a set of recommendations related to context, promptness, accuracy, resolution, and verification. Students commented that it is undeniable that ChatGPT’s capabilities have the potential to transform the programming process for developers. With its remarkable ability to generate multiple solutions to a problem and adapt to a developer’s preferred approach. Others pointed out how ChatGPT can eventually hamper the creativity of developers. Students anticipate that ChatGPT will revolutionize the way developers approach coding in the future. Other students felt that ChatGPT could quickly fix language-specific coding errors, along with PMD default issue descriptions, when they are well explained. But when the errors are propagated or occur in multiple locations, the students had difficulty querying the model. These limitations can be mitigated through formal training in prompt engineering. The fact that ChatGPT’s replies are determined by the prompt it gets and the data it was trained on could be a drawback as the quality of its responses are highly dependent on the quality of the prompts. Its solutions may not always be precise or appropriate for the particular activity or topic. The students also mentioned that ChatGPT is helpful in most situations. However, it struggles to solve issues without context. For example, ChatGPT might find the code comprehensive and easy to read, but when students mention the problem with the code, it then recognizes the issue. The more detailed the description, the more specific and accurate the solution made by ChatGPT. However, some issues require investigating various coupled files that may or may not be present as part of the input. Therefore, even after inputting a proper description of the buggy code, there are no guarantees on the model’s ability to debug. Aside from the detailed description of the issue, providing the most relevant code fragments is key to avoid dragging ChatGPT attention to either unnecessary details or bug-free code fragments.

**Summary for RQ<sub>3</sub>:** Overall, this assignment helps cultivate analytical and critical thinking skills as students engage in the debugging process. In addition, it teaches students to be skeptical towards the use of ChatGPT by shedding light on the limitations of ChatGPT in identifying and solving problems. It highlights the importance of incorporating other traditional static analysis tools and techniques to improve the accuracy and efficiency of their predictions. Furthermore, the involvement of a human-in-the-loop, capable of comprehending code, can be highly valuable and desirable.

## IV. REFLECTION

This section provides the lessons learned from both the educator’s and the student’s perspectives.

¶ **Lesson #1: Develop complementary assignments.** ChatGPT identified each of the PMD violations and attempted to fix them. In some cases, this would cause the problem to vanish completely from the PMD analyzer. In other cases,

Table II: Student's insight about the usefulness, usability and functionality of ChatGPT.

Theme	Sub-theme	Example (Excerpts from a related student's comment)
Usefulness	Automation	"The advantages of ChatGPT include its affordability, <b>capacity for automating tedious processes</b> , and ability to deliver precise and thorough responses for customer service departments."
	Quality	"PMD is a useful tool that can help identify potential issues in code and provide suggestions for improvement. It can also help enforce coding standards and best practices, <b>leading to cleaner, more maintainable code</b> . In my experience, both ChatGPT and PMD have been valuable tools. ChatGPT has been able to quickly provide helpful responses to various questions, while PMD has been able to catch potential issues that may have been missed during manual code reviews. However, it's important to note that these tools should be used as aids and not as replacements for human analysis and judgment. Overall, I believe that the use of AI language models and code analysis tools like PMD can greatly benefit developers and lead to better code quality and efficiency."
	Scope	"ChatGPT was very informative about giving context and definitions of errors, and was even able to provide definitions and recommendations specific to the PMD plugin. Negatives included when code errors were too vague or covered code segments that were too large for ChatGPT to understand. I conclude that <b>ChatGPT is useful for refactoring small segments of code</b> like fixing errors in smaller methods, nested if statements, and generally cleaning up the code to run faster, but not for larger issues like god classes."
	Awareness	"ChatGPT was extremely useful. <b>I had all of my answers I needed, and even was able to answer other questions for me</b> . When it fixed the code, it also gave explanations to each change it made to the code. Overall, ChatGPT was an extremely useful and easy to understand."
Usability	Experience	"The refactored code given by ChatGPT was bug free most of the time but at few times there were bugs in it. Overall the chatgpt is very nice application specially for those who are beginners in software development and programming."
	Documentation	"My experience with ChatGPT was largely positive. The tool was able to quickly identify a range of issues in the codebase, including code style issues, best practices, performance issues, and design problems. The reports generated by ChatGPT were very informative and easy to understand, providing clear explanations of each issue and suggesting steps to fix them."
	Ease of use	"It was quite easy to fix errors using ChatGPT. The trick is to find the right prompt so that it understands what it needs to do quickly rather than going around explaining the issue without providing the fix. However, since ChatGPT only had limited code visibility (the code snippet provided by the user), it generated a few fixes which would break the code (ex change variable names, use packages without importing, etc.). We can't say that the code fix provided by ChatGPT was entirely wrong though. Overall, it is easy and quick to use ChatGPT to fix most PMD Errors, with a little bit of oversight."
	Explanability	"when you're working on a small scale project it's extremely useful. Can also explain basic concepts you might have forgotten very well."
Functionality	User interface	"Chatgpt has a user-friendly interface. It provides precise and thorough responses for customer service teams. However, there are also potential disadvantages to using these models, such as the need to carefully adjust the model's output. It lacks originality and might give biased output."
	Correction	"ChatGPT would sometimes misunderstand what error I was trying to indicate, or would be confused by the fact that I was giving it a code excerpt rather than the full code, which was too large to provide to it. Its corrections sometimes also misunderstood the purpose of an excerpt and would slightly alter the logic, but this was expected."
	Detection	"Overall I had a positive experience with ChatGPT as it was able to detect the errors that PMD reported. However for the last 5 code segments I tested, it started generating code that produced compile errors and it didn't always generate the code in java on the first few attempts."
	Debugging	"ChatGPT is usually unable to identify the problems the pmd extension provides. However when it does it is able to go into great detail about the problems, and either fix the problems entirely or give guidance as to how to fix them. I enjoy using chatgpt for debugging and other things and I believe it is a great academic learning tool."
Recommendation	Context	"ChatGPT was definitely helpful to fix errors. However, many errors required a lot more context than what chatgpt accepts and that leads to incorrect fix or sometimes makes the problem worse. There is also the problem with it dropping context time and again. I feel chatgpt is best considered as a supplement to the existing options like stackoverflow but cannot be blindly trusted. That being said it is still a great upgrade from stackoverflow and may be in future be more accurate. Overall a positive experience."
	Promptness	"It was very useful, there was a bit of a learning curve as it took a few attempts to understand what exactly to prompt it to give you the information you want to know. I had used it before so that's why I understood how to problem solve in that way but chat GPT itself doesn't really suggest other ways to prompt if it doesn't get enough to give an answer."
	Accuracy	"GPT is not primarily designed for debugging but can assist in identifying syntax errors and offering debugging strategies. It can also suggest alternative solutions and offer insights into the program flow and potential issues. However, its suggestions may not always be accurate, and it may not have a deep understanding of complex code structures, making it less effective for identifying all types of errors."
	Resolution	"While ChatGPT was able to identify issues in the code, it did not always provide suggestions for how to fix them. This meant that developers still needed to have a good understanding of coding best practices in order to address the issues identified by ChatGPT."
	Verification	"ChatGPT is perfect as a code-companion but not as a replacement. The reason for this is because ChatGPT is unable to find context of the applications of code. This makes it more prone to missing logical bugs like the fall-through of a switch-case statement. Having a human-in-the-loop who is able to understand the applications and has a good idea of the overall structure of the project is much more desirable."

although a fix was applied, the PMD analyzer would still detect the problem as present. ChatGPT re-edits the code when prompted; sometimes, it will improve its response. However, when given the same prompt multiple times, it will generate the same response with alteration after many attempts. Therefore, students can use ChatGPT as a complementary tool alongside traditional static analysis tools to improve the efficiency of their software development process.

¶ **Lesson #2: Limited understanding of the broader context of the codebase.** While ChatGPT was informative about giving context and definitions of errors and could provide specific recommendations for PMD violations, it has a limited understanding of the broader context and the entire codebase, which could lead to missing dependencies and code dependences. Due to this limitation, there have been instances where ChatGPT makes suggestions based on misunderstandings or false assumptions about the code. This assignment reveals

this limitation in a practical manner to help students better understand the mechanics of the model, rather than treating it as a black box that autogenerates acceptable answers for any given query. In this experiment, not only students could not reach an acceptable answer for some queries, they also experienced how suggested code changes can be even problematic, as seen in Figure 6. Students experienced various negative side-effects of the autogenerated code. For example, in some cases, ChatGPT provided code has introduced compiler errors. Also, some students reported that the suggested refactoring did push some tests to fail, despite the fact that refactoring is supposed to preserve the system's internal behavior. Furthermore, ChatGPT suggested a fix for the 'Best Practices' category; but its solution has increased the cyclomatic complexity of the method, probably due to the lack of context and explainability of the code snippets. This observation is consistent with a previous study [6] that found that bad warning messages and

no suggested fixes are some of the pain points reported by industrial software developers when using program analyzers.

**¶ Lesson #3: Verify ChatGPT's responses with human expertise.** While ChatGPT can provide some help in debugging, it should not be relied on as the only debugging tool. ChatGPT is perfect as a code companion, but not a replacement. The reason for this is that ChatGPT cannot find the context of code applications. Having a human-in-the-loop who can understand the applications is much more desirable. Thus, experienced developers and specialized debugging tools remain essential for thorough and effective debugging.

**¶ Lesson #4: Automate the debugging and bugfixing process.** ChatGPT is an effective tool for programmers seeking assistance. Still, providing clear and detailed questions is essential to get the most accurate and helpful responses. It has helped students debug and clarify what each code segment did and explain how it performed each bugfix. Thus, the process is significantly shortened. Yet, in a few cases, it required some experimentation before coming up with a set of questions to accomplish the task. Additionally, it suggested changes that caused compiler errors, *e.g.*, renaming a variable in a field declaration or fixing a different issue in the code instead of the one given to analyze. Furthermore, ChatGPT tends to view PMD-related stylistic or best practice issues as not problematic and is too light handed in explaining the potential issues with stylistic/best practice problems in the source code. In the real world, this can lead to a student developing bad coding habits, ultimately detrimental to the productivity of the individual and the team they belong to. Some of the flaws were observed with inexplicable fixes or suggestions.

**¶ Lesson #5: Improve ChatGPT's accuracy and effectiveness.** ChatGPT was useful in solving a few PMD rule violations and provided good recommendations for resolving them. However, it cannot be determined as a PMD rule violation in a few cases, even though the code snippet is provided with context. For a few violations, it answered as 'may', providing it satisfying given conditions since it only provided the code snippet and not the surrounding code. We believe that if ChatGPT is trained properly for the specific ruleset, it can be used to quickly eliminate PMD rule violations. Yet, one cannot highly rely only on ChatGPT; some level of understanding and knowledge is required by the developer to make the appropriate correction, since it sometimes generates code that produces compile errors.

**¶ Lesson #6: Handle complex code and errors.** The effectiveness of ChatGPT is closely related to the quality of its training data set. According to the students' comments, ChatGPT provides high-level and valuable suggestions about the bugs. For instance, it is useful in fixing non-pressing matters such as code style, adding comments, making the if statements more logical, etc. On the other hand, it would sometimes misunderstand the error the student was trying to indicate or would be confused by the fact that the input gave it a code excerpt rather than the full code, which was too large to provide. Its corrections sometimes also misunderstood the purpose of an excerpt and would slightly alter the logic.

Therefore, the more complex the given code input, the more likely the code output will not work properly.

## V. RELATED WORK

Biswas [4] discussed an overview of ChatGPT as a language model developed by OpenAI that offers a wide range of computer programming capabilities. These capabilities include code completion, correction, prediction, error fixing, optimization, document generation, chatbot development, text-to-code generation, and technical query answering. The author highlighted the ability of ChatGPT to provide explanations and guidance to users and concluded that it is a powerful tool for the programming community. Haque and Li [11] explored the capabilities of ChatGPT as a debugging tool and the best practices for integrating it into the software development workflow. Their findings show that ChatGPT is a useful tool for debugging but should be used cautiously in software development. Ma *et al.* [15] performed a study evaluating ChatGPT capabilities and limitations in software engineering from three aspects: 1) syntax understanding, 2) static behavior understanding, and 3) dynamic behavior understanding. The authors concluded that ChatGPT possesses capabilities akin to an Abstract Syntax Tree (AST) parser, ChatGPT is susceptible to hallucination when interpreting code semantic structures and fabricating nonexistent facts, which underscore the need to explore methods for verifying the correctness of ChatGPT outputs to ensure its dependability in software engineering tasks.

## VI. THREATS TO VALIDITY

**External Validity.** Concerning the generalizability of our results, our study involves 102 submissions. Although we obtained valuable information and performed an accurate analysis, the results may not represent the larger student population that uses static analysis tools and ChatGPT to fix issues. Additionally, our analysis was performed on mature open-source Java projects that varied in size, contributors, and number of commits. However, we cannot claim the generality of our observations to projects written in other programming languages or belonging to different ecosystems. More research is needed on even more projects to mitigate this threat. Since ChatGPT training contains a large corpus of source code, from GitHub and StackOverflow, there is a significant chance of a data leakage problem, *i.e.*, the proposed fixes were previously seen in the training set, and so the fixes were previously memorized. However, the projects used actually contain some of these warnings in their current versions, which means that ChatGPT may have experienced these fixes in other projects, and its current decisions are based on inference. In addition, ChatGPT performance was not uniform across categories and underperformed for design-level issues. Finally, we recommend that students use the free version of ChatGPT (currently 3.5), but we did not control who uses the premium service, which features GPT-4.

**Internal and Construct Validity.** As for the completeness and correctness of our interpretation of open-ended comments

about the tool, we did not extensively discuss all comments because some of them are open to various interpretations and we need further follow-up interviews to clarify them. Additionally, to avoid personal bias during manual analysis, each step of the manual analysis was conducted by two authors until a consensus was reached. The choice of PMD, as a static analysis tool, may introduce some bias to the way these issues are detected, especially since the detection of bad programming practices and code smells is known to be subjective [5], [16], [7], [2], [19], [1]. Also, students may have had a different experience if another tool was selected in this assignment. We chose PMD as it is one of the popular state-of-the-art tools, but in future work, we plan on trying other static analysis tools to see if they can also reach this level of satisfaction. In addition, any training for students may induce training bias. To mitigate it, both PMD and ChatGPT training were independent. During ChatGPT training, students were exposed to prompts related to software quality principles, that they can relate to, but none of the queries was targeting the type of issues raised by PMD.

## VII. CONCLUSION AND FUTURE WORK

Understanding the practice of reviewing code to improve the quality is of paramount importance for education. Although modern code review is widely adopted in open-source and industrial projects, the relationship between using LLMs such as ChatGPT and how students perceive it during code analysis remains unexplored. In this study, we conducted a quantitative and qualitative study to explore the effectiveness of PMD and ChatGPT in familiarizing students with improving source code, by i) detecting code issues and antipatterns and ii) implementing fixes for their correction. The paper develops a culture of reviewing and patching unknown codes.

Our results reveal several types of static analysis tools that students should pay more attention to during code review; reviewing design-related changes takes longer to complete compared to other changes, and students rated some aspects of ChatGPT positively while also providing valuable ideas for future model improvement. For future work, we plan on using other static analysis tools that will complement and validate our current study to provide the software engineering community with a more comprehensive view of the use of static analysis tools to engage students with software quality improvement from the educator and student perspectives. Moreover, we plan to investigate students' understanding of code review practice using various real-world applications in a semester-long course project.

## REFERENCES

- [1] E. A. AlOmar, S. A. AlOmar, and M. W. Mkaouer. On the use of static analysis to engage students with software quality improvement: An experience with pmd. 2023.
- [2] F. Arcelli Fontana, M. V. Mäntylä, M. Zanoni, and A. Marino. Comparing and experimenting machine learning techniques for code smell detection. *Empirical Software Engineering*, 21(3):1143–1191, 2016.
- [3] G. Bavota, A. De Lucia, M. Di Penta, R. Oliveto, and F. Palomba. An experimental investigation on the innate relationship between quality and refactoring. *Journal of Systems and Software*, 107:1–14, 2015.
- [4] S. Biswas. Role of chatgpt in computer programming.: Chatgpt in computer programming. *Mesopotamian Journal of Computer Science*, 2023:8–16, 2023.
- [5] S. Bryton, F. B. e Abreu, and M. Monteiro. Reducing subjectivity in code smells detection: Experimenting with the long method. In *2010 Seventh International Conference on the Quality of Information and Communications Technology*, pages 337–342. IEEE, 2010.
- [6] M. Christakis and C. Bird. What developers want and need from program analysis: an empirical study. In *Proceedings of the 31st IEEE/ACM international conference on automated software engineering*, pages 332–343, 2016.
- [7] D. Di Nucci, F. Palomba, D. A. Tamburri, A. Serebrenik, and A. De Lucia. Detecting code smells using machine learning techniques: are we there yet? In *2018 ieee 25th international conference on software analysis, evolution and reengineering (saner)*, pages 612–621. IEEE, 2018.
- [8] L. Fei-Fei, R. Fergus, and P. Perona. One-shot learning of object categories. *IEEE transactions on pattern analysis and machine intelligence*, 28(4):594–611, 2006.
- [9] F. A. Fontana, P. Braione, and M. Zanoni. Automatic detection of bad smells in code: An experimental assessment. *J. Object Technol.*, 11(2):5–1, 2012.
- [10] R. M. French. Catastrophic forgetting in connectionist networks. *Trends in cognitive sciences*, 3(4):128–135, 1999.
- [11] M. A. Haque and S. Li. The potential use of chatgpt for debugging and bug fixing. *EAI Endorsed Transactions on AI and Robotics*, 2(1):e4–e4, 2023.
- [12] N. Imtiaz, B. Murphy, and L. Williams. How do developers act on static analysis alerts? an empirical study of coverity usage. In *2019 IEEE 30th International Symposium on Software Reliability Engineering (ISSRE)*, pages 323–333. IEEE, 2019.
- [13] B. A. Kitchenham and S. L. Pfleeger. Personal opinion surveys. In *Guide to advanced empirical software engineering*, pages 63–92. Springer, 2008.
- [14] C. Li, E. Chan, P. Denny, A. Luxton-Reilly, and E. Temporo. Towards a framework for teaching debugging. In *Proceedings of the Twenty-First Australasian Computing Education Conference*, pages 79–86, 2019.
- [15] W. Ma, S. Liu, W. Wang, Q. Hu, Y. Liu, C. Zhang, L. Nie, and Y. Liu. The scope of chatgpt in software engineering: A thorough investigation. *arXiv preprint arXiv:2305.12138*, 2023.
- [16] M. V. Mäntylä and C. Lassila. Subjective evaluation of software evolvability using code smells: An empirical study. *Empirical Software Engineering*, 11(3):395–431, 2006.
- [17] S. A. Mengel and V. Yerramilli. A case study of the static analysis of the quality of novice student programs. In *The proceedings of the thirtieth SIGCSE technical symposium on Computer science education*, pages 78–82, 1999.
- [18] M. M. Rahman and Y. Watanobe. Chatgpt for education and research: Opportunities, threats, and strategies. *Applied Sciences*, 13(9):5783, 2023.
- [19] D. Singh, V. R. Sekar, K. T. Stolee, and B. Johnson. Evaluating how static analysis tools can reduce code review effort. In *2017 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pages 101–105. IEEE, 2017.
- [20] R. Van Solingen, V. Basili, G. Caldiera, and H. D. Rombach. Goal question metric (gqm) approach. *Encyclopedia of software engineering*, 2002.
- [21] Y. Wang, Q. Yao, J. T. Kwok, and L. M. Ni. Generalizing from a few examples: A survey on few-shot learning. *ACM computing surveys (csur)*, 53(3):1–34, 2020.
- [22] J. White, S. Hays, Q. Fu, J. Spencer-Smith, and D. C. Schmidt. Chatgpt prompt patterns for improving code quality, refactoring, requirements elicitation, and software design. *arXiv preprint arXiv:2303.07839*, 2023.
- [23] Y. Xian, B. Schiele, and Z. Akata. Zero-shot learning—the good, the bad and the ugly. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4582–4591, 2017.
- [24] X. Yang, Z. Yu, J. Wang, and T. Menzies. Understanding static code warnings: An incremental ai approach. *Expert Systems with Applications*, 167:114134, 2021.
- [25] R. Yedida, H. J. Kang, H. Tu, X. Yang, D. Lo, and T. Menzies. How to find actionable static analysis warnings: A case study with findbugs. *IEEE Transactions on Software Engineering*, 2023.

# How Much Effort Do You Need to Expend on a Technical Interview? A Study of LeetCode Problem Solving Statistics

1<sup>st</sup> Jialin Cui

*Department of Computer Science  
North Carolina State University  
Raleigh, USA  
jcui9@ncsu.edu*

4<sup>th</sup> Ruochi Li

*Department of Computer Science  
North Carolina State University  
Raleigh, USA  
rli14@ncsu.edu*

2<sup>nd</sup> Runqiu Zhang

*Department of Statistics  
University of Virginia  
Charlottesville, USA  
rz2cv@virginia.edu*

3<sup>rd</sup> Fangtong Zhou

*Department of Computer Science  
North Carolina State University  
Raleigh, USA  
fzhou@ncsu.edu*

5<sup>th</sup> Yang Song

*Department of Computer Science  
University of North Carolina Wilmington  
Wilmington, USA  
songy@uncw.edu*

6<sup>th</sup> Ed Gehringer

*Department of Computer Science  
North Carolina State University  
Raleigh, USA  
efg@ncsu.edu*

**Abstract**—A technical interview is the culmination of the recruiting process for hiring software engineers in the tech industry. Many well-known companies, including Amazon, Meta (formerly Facebook), Alphabet (Google), and Microsoft, use it to filter candidates. However, the drawbacks of technical interviews are well-documented, including their lack of real-world relevance, bias towards newer developers, demanding time commitment, and potential to induce unnecessary anxiety and frustration. Despite these criticisms, there is no clear indication that the industry will alter the format of technical interviews in the near future. To assist student developers in preparing for these challenges, we conducted a quantitative analysis using over 300,000 user profiles from LeetCode, arguably the most popular online platform for preparing software development candidates for interviews. Our analysis aims to provide developers with insights into the effort required to prepare for technical interviews, especially in terms of solving programming questions, to secure a position at a renowned company.

**Index Terms**—LeetCode, career, hiring practices, software engineering, technical interviews, coding challenge

## I. INTRODUCTION

In 2022, the global IT professional services market was valued at USD 821.58 billion. It's projected to grow at a compound annual growth rate (CAGR) of 9.1% from 2023 to 2030 [1]. Even with this growth, getting a technical job remains challenging, especially at big companies like Amazon, Meta, and Google. One reason is the unique hiring process in the computing industry, which places a strong emphasis on technical interviews. These interviews assess a candidate's technical proficiency, understanding of computing, and problem-solving capabilities. Frequently, they feature live coding segments where candidates must articulate their thought process and devise optimal solutions focusing on efficiency and memory conservation [2]–[4]. As technical interviews are now common in the tech industry, developers need to know

how to prepare for them [2], [3], [5]. Some studies [6], [7] have pointed out issues with the current format of these interviews. Concerns include them not reflecting real-world tasks, having a bias against newer developers, taking up too much time, causing stress and frustration, and possibly overlooking good candidates. Yet, there's no sign that the tech industry will change how these interviews are conducted anytime soon.

In this study, rather than reiterating the common criticisms of technical interviews or discussing alternative formats proposed in prior research [6]–[8], we focus on understanding how programmers prepare for these challenges. Our aim is to guide emerging developers and computer science students in their endeavors. We analyzed 302,785 publicly available user profiles on LeetCode [9], a premier platform for software development interview preparation. We explored the correlation between users' problem-solving statistics and their global rating scores, seeking to understand which problem topics enhance problem-solving capabilities the most. Moreover, we created average profiles for several top-tier companies, using data from users who specified their employers. These profiles detail the number of problems typically tackled by an individual at a given company across 46 different types, as well as the programming languages they favor, covering 22 unique languages. Developers can use these average profiles as benchmarks against which they can assess their own readiness. Any discrepancies can highlight areas that may require further attention. Through this research, we provide a panoramic view of the preparation strategies programmers employ for technical interviews and equip developers with empirical insights to navigate technical interviews at elite companies.

## II. RELATED WORK

The field of technical interviews has not been extensively studied. However, we have identified some studies that have examined technical interviews and the associated preparations.

Lunn [10] investigated the experiences of undergraduate computing students with the hiring process by surveying 740 students across three universities. A noteworthy 52% confirmed undergoing technical interviews, and of them, approximately 9.43% expressed dissatisfaction, largely rooted in feelings of unpreparedness. Lunn postulated that academic institutions could perhaps bridge this gap by endorsing resources like LeetCode for student preparation.

Gant's book, "*Surviving the Whiteboard Interview*", likens the technical interview to a melange of conventional interviews, public speaking sessions, and challenging quizzes. Gant delved deep into the detrimental impact of stress on interviewees and subsequently proffered strategies to counter it [11].

In a compelling approach, Behroozi et al. [7], [12] incorporated eye-tracking tools and advanced computer vision algorithms to decipher interviewees' cognitive metrics. Their findings revealed an upswing in cognitive load and a dwindling attention span when participants attempted problem-solving on a whiteboard during an interview. The study led to recommendations on ways to alleviate these impediments.

Furthering their explorations, Behroozi and team [13] initiated a randomized controlled trial engaging 48 Computer Science students. Their objective was to discern the disparity in performance in solitary versus public problem-solving settings. The results were startling, indicating that an observing interviewer can slash performance rates by over half. This revelation underscored a pertinent concern: interviews might inadvertently sideline competent candidates by intertwining assessment with undue stress.

Another study spearheaded by Kapoor and Gardner McCune [14] engaged 536 CS undergraduates across three U.S. universities to glean insights about their internship pursuits. An intriguing 27.4% reported either securing internships or gearing up for them through technical interview drills on platforms like LeetCode [9], GeeksforGeeks [15], and HackerRank [16].

Lunn and colleagues [17] claimed that the expectation for applicants to prepare months or years before looking for a job neglects their current obligations and responsibilities. They conducted a survey in three public universities and found that white students began preparing earlier, spent more time preparing, and received more job offers than non-white students. Females spent more time preparing on average and received more job offers than non-female students.

Hall and Gosha [18] surveyed students about their experiences with technical interviews and found that interview performance decreased as interview anxiety increased. They also reported that mock interviews were commonly used to prepare, along with resources like LeetCode and the book *Cracking the Coding Interview* [3].

A subsequent study by Lunn, Zerbe, and Ross [19] aimed to pinpoint enhancements for the hiring process by gathering feedback from students who had previously faced technical interviews and bagged job offers. A recurring theme was the plea for universities to introduce dedicated courses to streamline their interview prep, with a pronounced nod to digital resources, prominently LeetCode.

Many studies aim to improve the technical interview process and enhance the experience for interviewees. However, there's a noticeable gap in guidance on effective preparation. Many students have mentioned using LeetCode as a preparation tool. With this in mind, we examined data from LeetCode. Our objective is to understand how experienced programmers prepare for technical interviews and, based on this, to create a guide for new developers and students. This guide will illustrate the amount of effort required to excel in technical interviews.

## III. BACKGROUND

In this section, we provide some necessary background information.

### A. LeetCode

LeetCode was founded in 2015 in Silicon Valley and rapidly expanded, establishing offices in both the U.S. and China. Its primary mission is to assist software engineers in enhancing their skills and advancing in their careers. Today, numerous developers turn to LeetCode to practice programming, deepen their understanding of algorithms, and prepare for job interviews.

The platform boasts an extensive library of over 2,000 programming challenges spread across more than 40 categories. Developers have the option to code in nearly 20 different programming languages. Furthermore, LeetCode offers integrated tools that allow users to test and debug their code directly on the site.

What sets LeetCode apart from other platforms, like Codeforces, is its emphasis on preparing individuals for real-world job interviews. While Codeforces is geared towards coding competitions, LeetCode offers problems akin to those encountered during actual interviews. Additionally, LeetCode has forums where developers can discuss their recent technical interview experiences, sharing insights about which problems to focus on and which ones require more practice. This community-driven approach has bolstered LeetCode's popularity, especially among emerging developers and university students.

A standout feature of LeetCode is its weekly coding contests. In these contests, global participants tackle four programming problems of varying difficulty levels within a span of one and a half hours. Their ranking is determined by the number of problems they solve and the time taken to address them. By participating, developers can gauge their performance against their peers worldwide. In a significant endorsement of its value, major companies like Amazon, Airwallex, and Motorola started partnering with LeetCode in 2021 to sponsor

these contests. This collaboration benefits both parties: while developers receive a platform to showcase their prowess, these tech giants are presented with a valuable recruitment avenue. This synergy underscores LeetCode's evolution from a mere practice platform to a crucial player in the tech recruitment landscape.

### B. Data Collection

We used LeetCode's API to create a script that extracts the user profiles of all participants who have taken part in any of the weekly LeetCode contests. By the time we executed our script in July 2023, there had been a total of 474 past contests.

An example of a typical LeetCode user's public profile is depicted in Fig. 1. The data shown in Fig. 1 can be readily accessed using the LeetCode API. For our study, we catalogued the different types of information outlined in Tables I and II for each user who had participated in at least one contest. The fields enclosed in the red box in Fig. 1 correspond to those in Table I, while the fields within the green box are detailed in Table II.

Table I displays the demographic and miscellaneous information recorded for each user, some of which require further explanation, as detailed in the table. The cell highlighted in blue is the key metric in our study. It represents the Elo rating [20], [21] for each user, calculated by the LeetCode ranking system based on their weekly contest performance.

The Elo rating system was initially introduced to rate the relative skill levels of players in games like chess. First adopted by the United States Chess Federation (USCF) in 1960, this system assigns an initial rating to each player, which is subsequently updated based on the outcomes of games played against other competitors. The difference in ratings between opponents, as well as the opponent's rating itself, determines the number of rating points gained or lost by a player after a game. For instance, if a higher-rated player defeats a lower-rated player, they will gain fewer rating points than if they had overcome a similarly-rated opponent. Conversely, a lower-rated player defeating a higher-rated opponent will earn more points than if they had bested someone of a similar rating.

Over time, as players continue to compete and their game outcomes vary, their Elo ratings change, representing their evolving skill levels compared to others in the system. The Elo rating system, now widely used across various games, sports, and esports, provides an objective measure of player skill, ensuring fair and competitive matchups between players of differing abilities.

In the context of our study, the Elo rating represents the user's relative ability to solve technical interview questions. A higher rating suggests a user is more adept at tackling such questions.

We recorded the number of programming questions each user solved in various programming languages. The first row of Table II displays all the programming languages supported by LeetCode that we recorded for each user. LeetCode categorizes its programming questions into three distinct levels: fundamental, intermediate, and advanced. The second, third, and

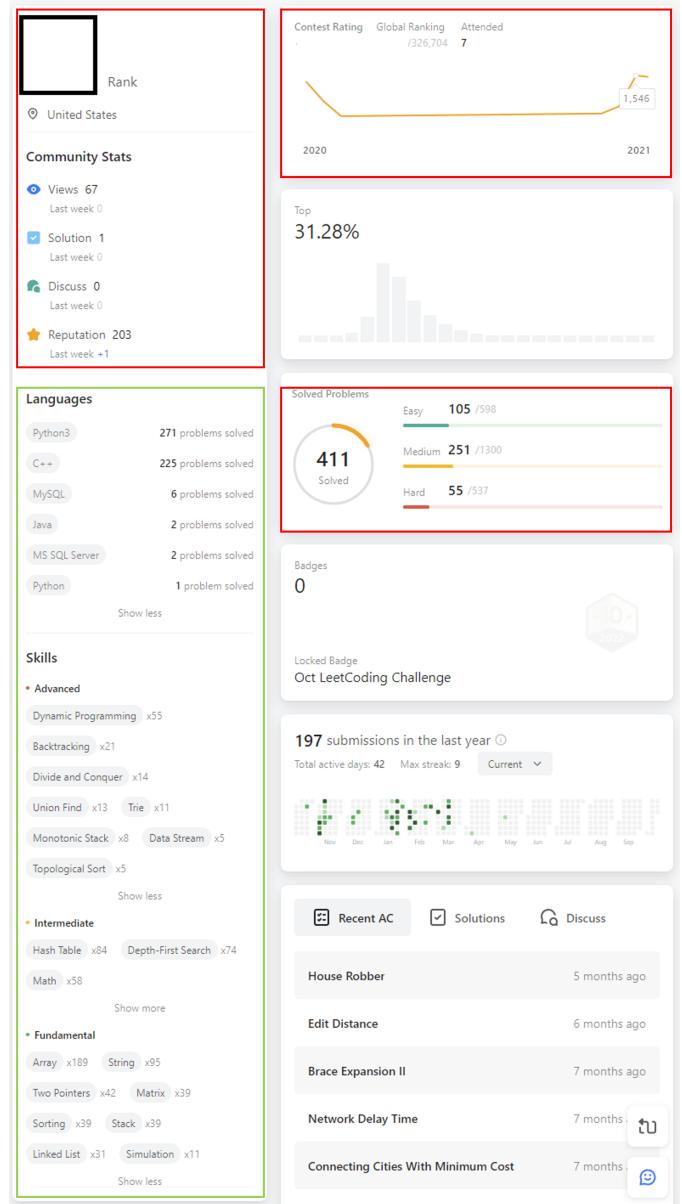


Fig. 1. A Typical User Profile From LeetCode

fourth rows of Table II list the problem topics corresponding to these levels, respectively. For each user, we also documented the number of programming questions they tackled within each topic. It's important to distinguish that the terms "fundamental," "intermediate," and "advanced" refer to problem topics, while "easy," "medium," and "hard" denote problem difficulties. For example, a problem under the "Array" topic is categorized at the fundamental level, yet its difficulty level could be labeled as "hard."

There are a few points worth mentioning. First, our data collection was limited to users who had participated in at least one weekly contest, given that only such users receive a rating and ranking from the website. As of mid-July 2023, our study collected 302,785 user profiles. While this seems like a

TABLE I  
MISCELLANEOUS INFORMATION ON LEETCODE USERS

Global ranking	The global ranking of this user based on his/her cumulative weekly contest rating calculated by LeetCode.
username	The user's username on LeetCode.
Problem ranking	The global ranking based on the total number of problems solved on LeetCode.
School*	The school that this user graduated from.
Country name*	The country that this user is living in.
Company*	The company that this user is working at.
Job title*	Job title of this user.
View count	The number of times other users viewed this user's post.
reputation	Reputation score calculated by LeetCode based on this user's activity.
Solution count	The number of solutions posted by this user.
Discuss count	The number of discussions this user participated in.
Contests count	The number of weekly contests this user participated in.
Rating	The Elo rating calculated by LeetCode based on this user's weekly contest performance.
Easy	The number of easy-level problems solved by this user.
Medium	The number of medium-level problems solved by this user.
Hard	The number of hard-level problems solved by this user.

Notes: The fields with \* are provided by the user. The user may choose not to provide this personal information.

substantial figure, it's essential to highlight that LeetCode's total user base is significantly larger. For context, LeetCode reported having over one million users back in 2017. They have not updated this official count since then, so the current total remains unknown. However, a noteworthy point is that LeetCode recorded 37.6 million visits in July 2023, two months prior to the drafting of this article [22], signifying

TABLE II  
LANGUAGES TYPES & PROBLEM TYPES

Languages	Erlang, Python, PHP, Racket, Bash, Oracle, C, C#, MS SQL Server, Java, Rust, Swift, MySQL, Ruby, Python3, Go, JavaScript, Scala, Elixir, C++, Kotlin, TypeScript
Fundamental	Array, Enumeration, Linked List, Matrix, Queue, Simulation, Sorting, Stack, String, Two Pointers
Intermediate	Binary Search, Binary Tree, Bit Manipulation, Brainteaser, Breadth-First Search, Database, Depth-First Search, Design, Graph, Greedy, Hash Table, Iterator, Math, Ordered Set, Randomized, Recursion, Sliding Window, Tree
Advanced	Backtracking, Binary Indexed Tree, Bitmask, Data Stream, Divide and Conquer, Dynamic Programming, Game Theory, Line Sweep, Monotonic Queue, Monotonic Stack, Quickselect, Rolling Hash, Segment Tree, Shortest Path, Suffix Array, Topological Sort, Trie, Union Find

its rapidly expanding user base.

Second, fields in Table I marked with \* are not mandatory for users to fill out; they provide this information voluntarily. As a case in point, when it comes to company affiliations used in our research, about 10.1% (or 30,627 out of 302,785) of users divulged which company they are associated with.

In summation, our finalized dataset encompasses global ratings and problem-solving statistics for 302,785 users, spanning 22 distinct programming languages and 46 varied problem topics. These users come from over 100 countries, are associated with around 1,000 companies, and have affiliations with approximately 5,000 educational institutions.

#### IV. METHODOLOGY

This section explains how we processed our raw data and the approach we used to conduct our analysis.

##### A. Data Processing

As previously mentioned, not all users disclosed their employer details. From the 302,785 users in our dataset, only 30,627 opted to share their company information, revealing affiliations with over 1,000 distinct companies. We scrutinized the provided company names and grouped users based on their reported workplaces. For further analysis, we chose 20 prominent tech companies, as detailed in Table III. It's worth noting that some renowned tech firms, such as Huawei, were omitted from our list because only a handful of users indicated they worked there. We deemed the sample size insufficient for analysis if fewer than 30 users claimed association with a particular company. While numerous companies had over 30 representatives within our dataset, we excluded them from our primary analysis (for instance, Infosys and Tata). Nonetheless, we're open to sharing data regarding specific companies upon reader interest. After narrowing down to the 20 companies, we aggregated users affiliated with the same company into a unified data frame to facilitate subsequent analyses.

##### B. Analysis Method

To comprehensively analyze our expansive dataset, which comprises 302,785 users, we employed a series of statistical tests. Primarily, we calculated the Pearson correlation coefficients, along with their respective p-values, to discern any linear relationships between the user ratings (highlighted in blue in Table I) and the number of problems they tackled in every programming language and problem category as denoted in Table II. Furthermore, we assessed the correlation between

TABLE III  
COMPANY NAME & NUMBER OF USERS

Amazon	1201	IBM	94	Apple	54
Google	796	Cisco	93	Salesforce	51
Microsoft	720	Paypal	82	Uber	49
Facebook	281	Adobe	78	Intel	47
Oracle	277	VMware	77	Nvidia	36
Samsung	212	Qualcomm	69	Bloomberg	35
Walmart	123	LinkedIn	60		

the ratings and the counts of easy, medium, and hard problems that users had addressed.

In our analysis, given that we are conducting multiple correlation tests, there's an increased risk of observing a significant  $p$ -value purely by chance. This could potentially lead to incorrect rejection of the null hypothesis. To counteract this, we apply the Bonferroni correction to adjust our  $p$ -values [23]. The essence of the Bonferroni correction is to adjust the confidence level from  $\alpha$  to  $\frac{\alpha}{n}$ , where  $n$  denotes the number of tests. Specifically, we adjusted each  $p$ -value by multiplying it by the number of correlation tests within a given set. If the adjusted  $p$ -value remains below 0.05, this indicates a statistically significant correlation. [24], [25]

Additionally, we compiled descriptive statistics for our user dataset. Specifically, for each metric under discussion for the 302,785 users, we calculated the average, standard deviation, as well as the 25th, 50th (median), and 75th percentiles. This data synthesis enabled us to derive an “average profile” that effectively epitomizes a typical LeetCode user.

To further enrich our analysis, for each of the 20 renowned tech companies outlined in Table III, we determined the average value for every aforementioned metric. This allowed us to create an “average company profile” for each firm, painting a portrait of the standard employee within these establishments. Such profiles can serve as invaluable benchmarks for aspiring job candidates, offering them a clearer picture of the proficiency levels typically exhibited by employees at these well-known tech companies.

## V. RESULTS

We present the calculated correlations in descending order in Table IV. All listed correlations are statistically significant ( $p < 0.01$ ), with the exception of the correlation between user rating and the number of problems solved in Oracle. This correlation is not statistically significant ( $p = 0.13$ ) and is therefore not included in the table. While some correlation coefficients are small (less than 0.3), all of them are statistically significant and positive.

Given the extensive number of metrics (70 different metrics), it's challenging to showcase every metric for the 20 companies in Table III. Hence, we've chosen the three most popular programming languages (C++, Python, and Java) and twenty problem topics that are commonly found in various computer science curricula. The average values for each of these metrics for every company are detailed in Tables V and VI. Table V displays the first ten companies, ranked in descending order based on their average rating, with Google (renowned for its rigorous technical interviews) taking the top spot. Conversely, Table VI lists companies ranked from eleventh to twentieth. The final column in both tables, shaded in light blue, presents the global average (derived from the 302,785 users in our dataset) for each metric, serving as a reference point. While the comprehensive table, encompassing all 70 metrics, is too expansive to be included in this paper, it can be accessed on figshare.com. Interested readers may view the complete table by following this link.

## VI. DISCUSSION

### A. Correlations between Problem Difficulty and Problem-Solving Ability Rating

In the red section of Table IV, we observe that users' problem-solving ability ratings correlate at 0.52 with the number of hard problems practiced, 0.47 with medium problems, and 0.30 with easy problems. This indicates that as a user's rating increases, they practice more hard problems relative to medium and easy ones. This suggests higher-rated developers tend to tackle more challenging problems than their lower-rated counterparts, which aligns with intuitive reasoning.

Given that none of the correlations are weak (with all  $r \geq 0.3$ ), it's advisable for developers to practice problems of all difficulty levels, as they all correlate with problem-solving ability ratings. Even though easy questions show the lowest correlation with problem-solving ratings, it doesn't mean they are insignificant. The prevalence of correctly answered easy questions among developers suggests that these questions might not differentiate between top-tier and average developers, but they remain fundamental. Ignoring hard-level problems due to their time-consuming nature or complex algorithms is not recommended, especially given their strong correlation with user ratings. Since easy problems are foundational and expected of any candidate, solely focusing on them won't differentiate a developer. This insight emphasizes the need for educators to integrate challenging topics into relevant university courses, better equipping students for technical interviews.

### B. Correlations Between Languages and Problem-solving Ability Rating

In the yellow section of Table IV, we see that the top five languages with the highest correlation coefficients to a user's overall rating are C++, Python3 (Python), Bash, MySQL, and Java. C++ has the top correlation at 0.33. This makes sense because C++ is a common choice in competitive programming due to its speed. This data suggests that high-rated developers often use C++ to tackle programming problems. In fact, C++ can solve some problems faster than other languages even with the same algorithm. Python3 is next with a correlation of 0.22. Because of Python's clear syntax and handy tools, it's arguably one of the easiest languages to use for problem-solving.

Bash and MySQL rank third and fourth. They aren't general-purpose languages but are crucial in specific settings, which might explain their higher correlations compared to other more universal languages. Java is fifth with a correlation of 0.09. Even though this isn't a strong correlation, it's still higher than most general-purpose languages. Since Java is widely taught in schools and universities, many developers are familiar with it. So, its correlation isn't as high, perhaps because regardless of a developer's skill, many might choose Java as their main language.

Other languages don't show a strong link with developer ratings (with  $r < 0.1$ ). This is understandable. While languages like C#, Go, and Ruby have their own benefits and user

TABLE IV  
CORRELATIONS BETWEEN PROBLEM-SOLVING RATINGS AND QUESTIONS SOLVED BY LANGUAGE AND PROBLEM TYPE

Hard	0.52	Ruby	0.02	Ordered Set	0.48	Math	0.45	Brainteaser	0.4
Medium	0.47	Scala	0.02	Binary Indexed Tree	0.48	Monotonic Queue	0.45	Data Stream	0.4
Easy	0.3	TypeScript	0.02	Bitmask	0.48	Rolling Hash	0.45	Tree	0.38
C++	0.33	Erlang	0.01	Topological Sort	0.48	Sorting	0.44	Divide and Conquer	0.38
Python3	0.22	PHP	0.01	Array	0.47	Breadth-First Search	0.44	Two Pointers	0.36
Python	0.11	Racket	0.01	Matrix	0.47	Backtracking	0.44	Binary Tree	0.36
Bash	0.1	C	0.01	Bit Manipulation	0.47	Trie	0.44	Randomized	0.36
MySQL	0.1	MS SQL Server	0.01	Sliding Window	0.47	Simulation	0.43	Line Sweep	0.36
Java	0.09	Swift	0.01	Segment Tree	0.47	Design	0.43	Recursion	0.35
Go	0.05	Elixir	0.01	Shortest Path	0.46	Monotonic Stack	0.43	Iterator	0.33
Rust	0.04	Greedy	0.5	Union Find	0.46	Depth-First Search	0.42	Quickselect	0.33
C#	0.03	Dynamic Programming	0.5	String	0.45	Queue	0.41	Linked List	0.31
JavaScript	0.03	Enumeration	0.48	Binary Search	0.45	Game Theory	0.41	Suffix Array	0.31
Kotlin	0.03	Graph	0.48	Hash Table	0.45	Stack	0.4	Database	0.1

Notes: This table contains the correlation coefficients between problem-solving ability ratings and the number of questions solved across different programming languages and problem types. Cells in red correspond to difficulty levels, cells in yellow correspond to language types, and cells in green correspond to problem types.

TABLE V  
AVERAGE COMPANY STATS: FIRST TO TENTH

	Google	Facebook	Bloomberg	LinkedIn	Nvidia	Uber	Amazon	Apple	Microsoft	Samsung	Global Avg
Rating	1,772.64	1,711.68	1,667.54	1,659.99	1,641.43	1,627.75	1,607.40	1,600.25	1,586.96	1,562.79	1,512.79
Contests	23.56	22.67	19.49	28.40	16.97	21.35	14.65	21.91	15.18	13.06	7.80
Easy	164.74	181.39	168.77	179.07	144.69	168.92	145.46	171.93	138.82	125.50	88.90
Medium	322.99	342.33	339.74	326.82	250.61	287.00	262.43	311.20	241.73	209.52	133.19
Hard	85.57	84.72	72.09	76.48	52.56	62.47	56.63	73.19	51.63	39.78	26.16
Java	195.76	177.39	176.09	275.77	38.28	226.76	191.62	241.83	150.57	84.97	76.14
Python3	125.23	183.67	224.54	117.23	135.14	129.35	92.89	186.87	90.30	40.14	50.20
C++	237.55	180.98	180.23	159.27	261.89	97.80	173.40	135.69	156.63	256.37	115.95
String	142.22	154.42	141.77	145.65	114.08	124.57	111.74	139.85	105.48	84.87	57.40
Array	294.74	308.82	294.46	294.23	225.97	263.69	236.52	285.93	217.62	192.06	125.96
Linked List	28.30	30.43	31.91	31.00	23.42	26.61	26.53	29.41	25.26	22.72	15.65
Hash Table	113.39	122.72	121.46	115.90	89.28	103.14	92.01	114.26	86.64	72.45	48.44
Stack	38.02	41.20	39.63	40.92	30.47	34.65	32.00	38.70	29.94	25.15	16.73
Queue	9.01	9.67	8.83	9.88	6.53	8.00	7.16	9.22	6.70	5.61	3.58
Tree	65.43	70.84	70.11	71.37	52.89	62.41	58.13	67.52	54.60	46.80	29.76
Binary Tree	59.32	64.37	64.40	65.32	48.78	56.84	53.41	61.57	50.14	42.79	27.46
Sorting	68.75	72.03	71.51	67.98	52.58	63.02	56.61	68.80	52.16	47.85	30.76
Binary Search	48.05	51.00	47.83	48.37	34.92	43.51	38.97	46.41	36.06	32.50	21.40
Graph	22.94	23.66	23.29	22.60	15.11	19.61	17.99	20.91	16.28	15.33	8.89
DFS	81.20	87.64	87.20	86.35	62.42	74.88	69.55	82.22	64.83	55.22	34.69
BFS	60.87	65.39	66.14	63.50	45.31	55.80	51.59	60.19	47.49	41.36	25.81
Math	86.10	90.45	81.97	85.12	66.31	72.96	65.46	82.76	61.77	51.97	35.84
Two Pointers	54.45	59.21	57.86	56.98	44.61	52.12	47.48	56.89	45.27	39.46	27.93
Greedy	53.30	55.17	51.29	50.95	39.06	46.63	41.67	49.17	37.28	35.02	21.18
Bit Manipulation	32.14	32.81	31.20	31.10	25.17	26.33	24.56	30.96	23.28	20.36	13.24
DP	93.83	91.82	85.94	87.70	67.56	77.84	73.28	83.07	66.38	59.11	38.00
Backtracking	26.32	28.43	27.09	26.57	20.83	22.39	20.78	25.80	19.47	15.57	10.67
DAC	15.60	16.40	16.74	16.75	13.33	14.33	14.06	15.94	13.25	11.78	8.13

Notes: DFS: Depth-First Search, BFS: Breadth-First Search, DP: Dynamic Programming, DAC: Divide and Conquer

groups, they don't have as large a user base as C++, Java, or Python3. They are useful and liked, but we didn't find a strong connection between a developer's problem-solving skills and the use of these languages.

To sum it up, for interview prep, especially for beginners, C++, Python3, and Java are better bets than other general-purpose languages. For those with some background, C++ stands out since top-rated users seem to prefer it. But just using C++ doesn't mean someone is a better developer. If C++ isn't for you, Python3, with its simple syntax, is a good alternative. And for those sticking to what they learned in school, Java remains a solid option.

### C. Correlations Between Problem Types and Problem-solving Ability Rating

In the green section of Table IV, the correlations between the number of each type of problem solved and problem-solving ability ratings are displayed. They're sorted in descending order from left to right and top to bottom. We notice that, aside from database problems ( $r = 0.10$ ), all other problem types have moderate-level correlations ( $> 0.30$ ) with user ratings. Within the top ten, several challenges often considered the "deadly foes" of developers stand out. For example, dynamic programming (DP) and greedy algorithms both have a correlation of 0.5, making them the top challenges.

TABLE VI  
AVERAGE COMPANY STATS: ELEVENTH TO TWENTIETH

	Adobe	Intel	Salesforce	VMware	Oracle	Cisco	Qualcomm	Paypal	IBM	Walmart	Global Avg
Rating	1,542.15	1,531.50	1,517.65	1,510.17	1,508.10	1,496.58	1,491.06	1,486.67	1,473.53	1,470.41	1,512.79
Contests	13.05	13.79	11.02	18.38	11.76	11.58	9.61	9.89	9.98	14.15	7.80
Easy	138.58	137.91	139.41	148.74	132.98	144.15	116.12	133.73	107.91	144.29	88.90
Medium	239.04	193.04	226.20	227.57	214.35	196.84	168.48	219.18	158.14	230.59	133.19
Hard	47.60	31.62	42.31	41.14	39.22	34.59	27.67	42.87	30.11	44.72	26.16
Java	162.48	87.79	252.61	225.34	185.52	171.88	59.65	230.83	87.87	240.55	76.14
Python3	46.09	102.57	43.88	63.84	51.47	91.19	48.80	63.79	80.55	57.24	50.20
C++	231.51	152.34	135.35	109.01	142.79	105.28	210.88	96.84	119.28	87.58	115.95
String	101.60	87.06	101.63	99.17	90.20	90.88	72.29	93.23	69.59	97.71	57.40
Array	214.23	182.53	205.92	209.01	196.39	191.20	161.35	196.99	152.36	210.33	125.96
Linked List	25.29	21.89	25.59	24.34	23.47	23.53	22.36	24.84	18.33	24.77	15.65
Hash Table	83.21	72.70	85.29	84.13	76.18	74.48	60.87	77.87	59.60	81.25	48.44
Stack	29.75	23.94	28.78	29.19	26.68	25.19	22.62	27.95	19.63	28.28	16.73
Queue	6.64	5.49	5.98	6.44	5.73	5.56	4.84	6.24	4.29	6.17	3.58
Tree	53.97	45.66	55.16	54.99	49.86	45.39	40.28	51.10	33.73	53.06	29.76
Binary Tree	49.65	42.34	50.75	50.64	45.76	42.26	37.67	47.13	30.70	49.01	27.46
Sorting	51.51	44.77	49.86	50.86	47.45	47.32	39.57	47.43	37.71	50.90	30.76
Binary Search	36.57	29.26	33.88	35.08	32.79	30.54	26.00	31.96	26.88	35.44	21.40
Graph	16.34	11.89	15.71	14.75	14.69	12.56	10.41	14.98	10.62	15.62	8.89
DFS	63.55	53.11	65.69	63.29	58.26	52.68	46.26	60.15	39.72	61.74	34.69
BFS	47.40	38.81	48.71	45.79	42.90	38.49	33.72	45.01	29.12	45.89	25.81
Math	58.95	52.94	57.69	59.96	53.50	53.80	45.46	52.33	41.48	57.38	35.84
Two Pointers	44.94	38.83	44.86	44.52	41.02	41.85	38.35	42.80	33.38	44.75	27.93
Greedy	37.48	30.28	33.63	35.22	33.92	30.58	26.68	33.68	26.37	36.82	21.18
Bit Manipulation	22.78	20.79	21.51	22.75	20.19	20.27	18.70	20.38	15.57	21.97	13.24
DP	68.06	48.32	62.53	59.97	58.96	52.46	48.61	60.91	41.81	63.86	38.00
Backtracking	18.00	14.89	18.25	18.03	16.45	15.08	13.93	16.88	11.47	18.21	10.67
DAC	12.74	11.30	12.39	12.77	11.91	12.14	10.83	12.21	8.88	12.92	8.13

Notes: DFS: Depth-First Search, BFS: Breadth-First Search, DP: Dynamic Programming, DAC: Divide and Conquer

Technical interviews frequently test dynamic programming skills to gauge developers' ability to address problems holistically. Greedy algorithms, however, assess if developers can identify a locally optimal solution. As such, struggling with a DP or greedy problem often leads to the failure of the technical interview.

Further, topics like enumeration, graph problems, ordered set, binary indexed tree, bitmask, and topological sort all exhibit a correlation coefficient of  $r = 0.48$ . These are foundational subjects in university algorithm and data structure courses. Basic computer science topics like arrays ( $r = 0.47$ ), strings ( $r = 0.45$ ), and sorting ( $r = 0.44$ ) also show a significant correlation with problem-solving ratings.

This data suggests that to do well in technical interviews, developers should thoroughly understand a range of computer science subjects. There aren't shortcuts. While some topics (like DP, greedy algorithms, and graphs) may be slightly more crucial than others, it's unwise to neglect any area. It's more beneficial to improve across all subjects than to miss out on a key topic.

Our analysis indicates that modern computer science students must prepare extensively to excel in technical interviews. This suggests that educators should emphasize topics with higher correlation coefficients. While university algorithms and data structure courses typically cover subjects like arrays and graphs effectively, other topics, such as dynamic programming or greedy algorithms, might be reserved for advanced courses. Educators should consider introducing these to undergraduate courses for better student preparation. Additionally,

topics like combinatorics, which might not be a focus in many courses, should receive more attention given its importance in our findings. We also recommend that instructors incorporate more coding problems in their classes, rather than just sticking to theoretical concepts.

#### D. Company Stats and Global Average

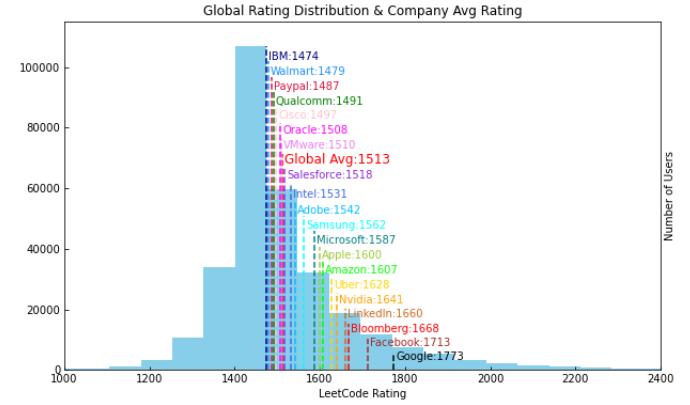


Fig. 2. Global Rating Distribution and Companies' Average Rating

Fig. 2 shows the distribution of global ratings and the average rating of each company in our dataset. As mentioned earlier, we calculated the average rating for each company by averaging the ratings of all users who reported working for that company. The red line represents the global average rating of

1513. Clearly, 13 out of the 20 tech companies in our dataset exceed the global average in terms of rating.

The top ten companies with the highest average ratings are Google (1773), Facebook (1713), Bloomberg (1668), LinkedIn (1660), Nvidia (1660), Uber (1628), Amazon (1607), Apple (1600), Microsoft (1587), and Samsung (1562). This ranking makes sense since companies like Google and Facebook are known for their challenging technical interviews. They are closely followed by firms like Bloomberg and LinkedIn, which are also reputed for their tough interview processes. Additionally, we observed that hardware and backend-focused companies such as IBM (1474), Qualcomm (1491), and Cisco (1497) have relatively lower average ratings. Our data suggest that aspiring employees aiming for hardware-oriented companies might not need to invest as much effort in problem-solving practice as those targeting firms more focused on software.

In conclusion, a company's average rating can offer insight into the rigor of its technical interviews. Developers can use this rating as a benchmark to gauge the difficulty of securing a job at a particular company. Since we derive a company's average rating from its employees, it should mirror the average capabilities of its staff. If a developer seeks to work with such colleagues, it's advisable to match their skill level. For instance, if a developer's LeetCode rating is 1500, below the global average of 1519, and they're aiming for Google with an average of 1773, they might find the interview quite challenging. Our data provides a robust reference for the level of preparation required. Instead of just identifying which companies conduct tough interviews, our research delivers quantifiable insights that can be more easily interpreted by developers and readers.

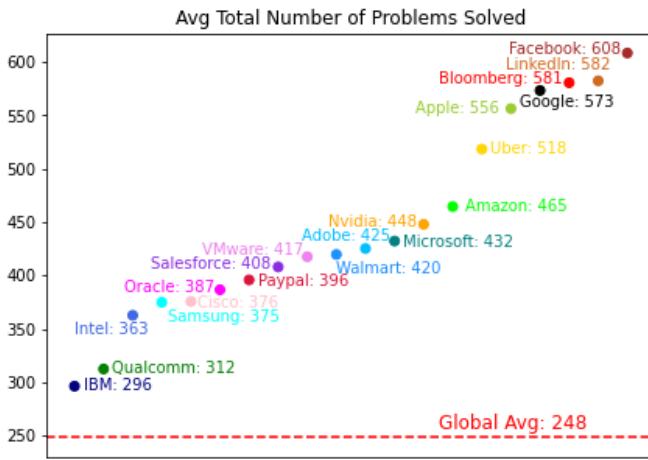


Fig. 3. Average Number of Problems Solved, by Company

Figure 3 displays the average number of problems solved by each company's employees. This number is derived by summing the Easy, Medium, and Hard fields from Tables V and VI for a specific company. The numbers are sorted in ascending order from bottom left to top right. The horizontal red line marks the global average. This suggests that a typical LeetCode user who has participated in at least one weekly con-

test has solved 88.9 easy problems, 133.2 medium problems, and 26.2 hard problems, amounting to a total of 248 problems. It's important to note that this average only pertains to users who've participated in contests. The overall average across all LeetCode users would be lower.

From Figure 3, it's evident that six companies boast an average problem-solving count exceeding 500. These companies are Uber (518), Apple (556), Google (573), Bloomberg (581), LinkedIn (582), and Facebook (608). If developers find the ratings in Figure 2 somewhat abstract, then Figure 3 offers a tangible benchmark, representing the average problem-solving effort of employees in each company. Developers can consult Tables V and VI to gauge how their own problem-solving numbers stack up against those of their desired company. If a developer's count exceeds a company's average, they can approach interviews with more confidence. Conversely, if their numbers are significantly lower, they should recognize the need for more practice. Ultimately, our data aims to provide developers with a clear reference point for the preparation required for technical interviews.

Lastly, Figure 4 displays the average number of problems each company solved using C++, Python3, and Java. The red line signifies the global average for problems tackled in these languages. These numbers ascend from the bottom left to the top right. A notable variance in language preference is evident across different companies.

For instance, the left side of the figure shows LinkedIn employees predominantly using Java. This is consistent with LinkedIn's reliance on Java for its platform and its recruitment of skilled Java developers. On the right side, Nvidia, Samsung, and Google employees have tackled the most problems using C++. Given these companies' extensive use of C++, it's logical they seek experts in this language. When it comes to Python3, Bloomberg leads, reflecting Python's status as a primary language at Bloomberg.

This visualization offers developers insights into which programming languages to emphasize during their technical interview preparations, contingent upon their target company. For instance, for those eyeing Google, honing their C++ skills seems judicious. However, it's essential to understand this as a general trend; one can secure a position at these firms without necessarily mastering their most-utilized language.

Both Table V and Table VI enumerate the variety and quantity of problems each company addresses. Developers can leverage this data to discern which problems warrant more attention. For instance, the average Google developer tackled 93.8 dynamic programming challenges, contrasted with just 22.9 graph-related problems. A hopeful Google interviewee might consider prioritizing dynamic programming challenges for their preparation.

These insights are particularly beneficial for developers, especially computer science students, who face time constraints in their interview preparations. Practicing every kind of problem is a tall order, so prioritizing those frequently addressed by successful candidates of a desired company is a pragmatic approach.

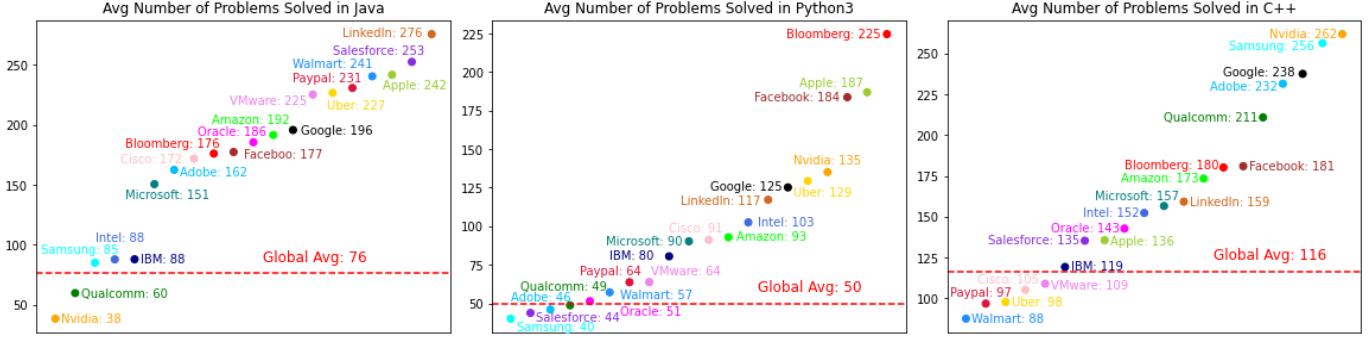


Fig. 4. Average Number of Problems Solved in Java, Python3, and C++ by Company

### E. Implications

Our study holds significance for both developers and computer science educators. Developers can harness our analysis to choose a programming language that aligns with the preferences of their target companies. Furthermore, they can gauge the gap between their problem-solving abilities and the average competence of employees at their aspirational company, pinpointing which challenges require more practice. Our data equips developers with a tangible benchmark to gauge the effort they should invest.

For computer science educators, our findings can guide curricular adjustments to better ready their students for technical interviews. Given that the typical LeetCode user practices around 248 problems, educators might consider introducing platforms like LeetCode in introductory courses. Additionally, they can structure algorithm and data structure classes around topics frequently tackled by seasoned industry professionals.

We are confident that our research can offer invaluable insights for both budding professionals and their mentors.

### VII. THREATS TO VALIDITY

The primary threat to the validity of our study comes from the data we collected. Since not all users provided their company information, we cannot get a complete picture of every company. The conclusions we draw are from what we have, but it does reflect the truth to a certain degree. Seocnd, correlation does not imply causation. Just because two things are reasonably well correlated does not mean that there is necessarily a causal relationship between them. It is possible that there might be post-hire organizational incentives to participate that could skew the data in ways that make it misleading to prospective interviewees.

### VIII. CONCLUSIONS AND FUTURE WORK

Our detailed analysis of LeetCode user data provides a clear picture of developers' problem-solving abilities. We've identified correlations with factors such as preferred programming languages and types of problems tackled.

By analyzing the average profiles of 20 leading tech companies, we've established a concrete benchmark for job seekers. This data offers developers a valuable perspective, enabling

them to compare their skills and accomplishments with those of professionals in top-tier companies.

This study benefits both developers and educators. Developers can gain insights that guide their decisions in selecting programming languages and prioritizing which problems to practice. This prepares them better for technical interviews. On the other hand, educators can adjust their teaching methods and content to better reflect the real-world needs of the tech industry. This approach not only narrows the gap between academia and the industry but also enhances employment opportunities for new graduates.

In summary, our research provides an in-depth view of the current state of tech recruitment and offers guidance for future developments. This ensures that both developers and educators stay updated and prepared for the ever-changing tech landscape.

Future studies could include a combined analysis of LeetCode and GitHub data. As suggested by numerous previous studies [26]–[31], GitHub user data can reflect various aspects of a developer's profile. It would be promising to analyze the correlation between users' performance on LeetCode and their contributions on GitHub to determine if there are any connections.

## REFERENCES

- [1] www.grandviewresearch.com, “It professional services market size, share & trends analysis report by type (project-oriented services, ito services, it support & training services), by deployment, by enterprise size, by end-use, by region, and segment forecasts, 2023 - 2030.” Mar. 2021. [Online]. Available: <https://www.grandviewresearch.com/industry-analysis/it-professional-services-market#:~:text=Report%20Overview,9.1%25%20from%202023%20to%202030>.
- [2] S. J. Lunn and M. S. Ross, “Cracks in the foundation: Issues with diversity and the hiring process in computing fields,” in *2021 ASEE Virtual Annual Conference Content Access*. Virtual Conference: ASEE Conferences, 2021.
- [3] G. L. McDowell, *Cracking the coding interview: 189 programming questions and solutions*. CareerCup, LLC, 2015.
- [4] M. Wyrich, D. Graziotin, and S. Wagner, “A theory on individual characteristics of successful coding challenge solvers,” *PeerJ Computer Science*, vol. 5, p. e173, 2019.
- [5] W. Gant, “Why software development interviews are hard,” in *Surviving the Whiteboard Interview*. Springer, 2019, pp. 1–6.
- [6] M. Behroozi, C. Parnin, and T. Barik, “Hiring is broken: What do developers say about technical interviews?” in *2019 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, IEEE, Roma, Italy: IEEE, 2019, pp. 1–9.
- [7] M. Behroozi, S. Shirodkar, T. Barik, and C. Parnin, “Debugging hiring: What went right and what went wrong in the technical interview process,” in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Software Engineering in Society*, 2020, pp. 71–80.
- [8] M. Behroozi, “Toward fixing bad practices in software engineering hiring process,” Ph.D. dissertation, North Carolina State University, 2022.
- [9] leetcode.com, “Leetcode - the world’s leading online programming learning platform,” Oct. 2022. [Online]. Available: <https://leetcode.com>
- [10] S. Lunn, M. Ross, Z. Hazari, M. A. Weiss, M. Georgopoulos, and K. Christensen, “The impact of technical interviews, and other professional and cultural experiences on students’ computing identity,” in *Proceedings of the 26th ACM Conference on Innovation and Technology in Computer Science Education V. 1*, 2021, pp. 415–421.
- [11] W. Gant, “Why software development interviews are hard,” in *Surviving the Whiteboard Interview*. Springer, 2019, pp. 1–6.
- [12] M. Behroozi, A. Lui, I. Moore, D. Ford, and C. Parnin, “Dazed: measuring the cognitive load of solving technical interview problems at the whiteboard,” in *Proceedings of the 40th International Conference on Software Engineering: New Ideas and Emerging Results*, 2018, pp. 93–96.
- [13] M. Behroozi, S. Shirodkar, T. Barik, and C. Parnin, “Does stress impact technical interview performance?” in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2020, pp. 481–492.
- [14] A. Kapoor and C. Gardner-McCune, “Exploring the participation of cs undergraduate students in industry internships,” in *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*, 2020, pp. 1103–1109.
- [15] “Geeksforgeeks - a computer science portal for geeks,” <https://www.geeksforgeeks.org/>, Oct. 2022.
- [16] “Hackerrank - market-leading technical assessment and remote interview solution for hiring developers,” <https://www.hackerrank.com/>, Oct. 2022.
- [17] S. J. Lunn, M. S. Ross, Z. Hazari, M. A. Weiss, M. Georgopoulos, K. Christensen, and T. Solis, “Uneven playing field: Examining preparation for technical interviews in computing and the role of cultural experiences,” in *2021 ASEE Virtual Annual Conference Content Access*, 2021.
- [18] P. Hall Jr and K. Gosha, “The effects of anxiety and preparation on performance in technical interviews for hbcu computer science majors,” in *Proceedings of the 2018 ACM SIGMIS Conference on Computers and People Research*, 2018, pp. 64–69.
- [19] S. J. Lunn, E. Zerbe, and M. S. Ross, “Need for change: How interview preparation and the hiring process in computing can be made more equitable,” in *2022 CoNECD (Collaborative Network for Engineering & Computing Diversity)*, 2022.
- [20] A. E. Elo, “The proposed uscf rating system. its development, theory, and applications,” *Chess Life*, vol. 22, no. 8, pp. 242–247, 1967.
- [21] A. Elo, *The Rating of Chessplayers, Past and Present*. Arco Pub, 1978.
- [22] SEMrush, “Leetcode.com - semrush overview,” <https://www.semrush.com/website/leetcode.com/overview/>, accessed 2023.
- [23] W. Research, “Bonferroni correction,” <https://mathworld.wolfram.com/BonferroniCorrection.html>, accessed on Janu 23, 2023.
- [24] H. Akoğlu, “User’s guide to correlation coefficients,” *Turkish Journal of Emergency Medicine*, vol. 18, pp. 91 – 93, 2018.
- [25] W. R. Rice, “Analyzing tables of statistical tests,” *Evolution*, vol. 43, no. 1, pp. 223–225, 1989.
- [26] J. Cui, R. Zhang, R. Li, Y. Song, F. Zhou, and E. Gehring, “Correlating students’ class performance based on github metrics: A statistical study,” in *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1*, ser. ITiCSE 2023. New York, NY, USA: Association for Computing Machinery, 2023, p. 526–532.
- [27] J. Cui, R. Li, K. Lou, C. Liu, Y. Xiao, Q. Jia, E. Gehring, and R. Zhang, “Can pre-class github contributions predict success by student teams?” in *Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: Software Engineering Education and Training*, ser. ICSE-SEET ’22. New York, NY, USA: Association for Computing Machinery, 2022, p. 40–49. [Online]. Available: <https://doi.org/10.1145/3510456.3514144>
- [28] J. Cui, R. Zhang, R. Li, F. Zhou, Y. Song, and E. Gehring, “How pre-class programming experience influences students’ contribution to their team project: A statistical study,” in *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1*, ser. SIGCSE 2024. New York, NY, USA: Association for Computing Machinery, 2024, p. 255–261. [Online]. Available: <https://doi.org/10.1145/3626252.3630870>
- [29] J. Cui, F. Zhou, R. Zhang, R. Li, C. Liu, and E. Gehring, “Predicting students’ software engineering class performance with machine learning and pre-class github metrics,” in *2023 IEEE Frontiers in Education Conference (FIE)*, 2023, pp. 1–9.
- [30] J. Cui, R. Zhang, R. Li, F. Zhou, Y. Song, and E. Gehring, “A comparative analysis of github contributions before and after an oss based software engineering class,” in *Proceedings of the 2024 Innovation and Technology in Computer Science Education V. 1*, ser. ITiCSE 2024, 2024.
- [31] J. Cui, F. Zhou, C. Liu, Q. Jia, S. Yang, and E. Gehring, “Utilizing the constrained k-means algorithm and pre-class github contribution statistics for forming student teams,” in *Proceedings of the 2024 Innovation and Technology in Computer Science Education V. 1*, ser. ITiCSE 2024, 2024.

# Leveraging Peer-assessment in Project-based Software Engineering Courses

Haneen Hijazi  
Department of Software Engineering  
The Hashemite University  
Zarqa, Jordan  
haneen@hu.edu.jo

Yasser Ali Alshehri  
Lane Department of Computer Science and Electrical Engineering  
West Virginia University  
virginia, USA  
yaalshehri@mail.wvu.edu

**Abstract**— Higher educational institutions seek to improve the quality and the productivity of the educational process. The current attitude is toward involving students in the learning and evaluation process. Peer review has been proven to be one of the most effective tactics to attain this in software engineering disciplines wherein project-based courses are substantial to afford high quality competencies. A few studies in literature empirically studied the impact of peer review in project-based software engineering courses. In this work, we attempt to provide more insights by implementing peer assessment in one of the project-based courses offered by the Department of Software Engineering at the Hashemite University, Jordan, that is “Object-Oriented Software Development”. In this paper, we investigate the validity of peer assessment by examining how well students in this course evaluate their peers and how the strength of students affects their assessment. The work also embeds a rubric that comprises key criteria to assess software system modelling. The results of the study reveal promising signs of using peer assessment in project-based software engineering courses.

**Keywords**— *peer-assessment, peer evaluation, peer-review, software engineering education, UML, rubric, project-based courses*

## I. INTRODUCTION

Modern direction in Higher Education Institutions (HEI) worldwide aims at improving the involvement of students in the educational process. Not only in the learning delivery but also in the assessment and evaluation. One way to achieve this is to leverage peer assessment. Peer assessment is a set of activities that enables students to evaluate and provide feedback on the work of their colleagues [1]. Through peer assessment, students will learn from other solutions, mistakes, and benefit from their peers' feedback. On the short term, peer assessment helps students get a faster and probably several feedback from different sources, allows better understanding of instructor assessment, improves students' assessment skills, fosters self-evaluation [2], promotes “assessment as learning” and self-regulated learning. On the long term, peer assessment promotes students'-centred learning and develops several transferable skills such as critical thinking, problem solving, analytical reasoning, self-management, teamwork, and communication skills [3].

Students of Information and Communication Technology (ICT) disciplines, including software engineering, must acquire practical skills to compete in the labor market. These competencies are normally assessed through practical projects assignments. Hence, project-based Learning was introduced. Project-based learning (PBL) is an approach in software engineering education that allows collaborative learning wherein students work in groups to build a software system in an educational context [4]. Several courses offered by the department of software engineering at the Hashemite

University employ this approach. One course is the “Object-Oriented Software Development” (OOSDEV); a second-year course that concentrates on object-oriented modelling using Unified Modelling Language (UML).

Modelling is an important activity in software engineering projects. Therefore, many courses in software engineering (including OOSDEV) employ modelling. In the case of the (OOSDEV) course, it is designed to be a prerequisite course for other several courses in the department, hence, peer assessment becomes highly beneficial as there are still opportunities to utilize the evaluation feedback to improve students learning and leveraging them in other future courses.

During the course, a group project is assigned to students that asks them to model a specific software system using Unified Modelling Language (UML). However, Software Engineering students face the challenging task of gaining the technical skills required to model software systems. Moreover, evaluating these projects is critical for students and somehow a burden task on the instructor. In order to enhance the learning of modelling using UML and to facilitate the assessment process, we introduce peer assessment in evaluating the course project. Peer assessment in software engineering project courses helps instructors assess teams and enhances the learning experience of students [4]. This form of assessment can be considered as “assessment as learning” as students are engaged in the assessment process, and they also learn while doing the assessment. Thus, peer assessment would help students get faster and probably several pieces of feedback, understand the assessment of the instructor and how to assess with an ultimate goal to promote student's centered learning, transferrable, and lifelong skills, which are required skills for a professional software engineer [5].

In this paper, we investigate the validity of peer assessment in evaluating modelling software engineering projects. Hence, this research aims at answering the following research questions:

- RQ1: What are the main criteria to be considered in assessing UML projects?
- RQ2: How well students in (OOSDEV) course evaluate each other?
- RQ3: How students' strength affects their peer assessment?

To answer the previous questions, peer assessment of a single instance (OOSDEV) course (fall 2023-2024) was implemented, and results were analyzed.

The rest of this article is organized as follows: Section II presents previous work related to current research in terms of peer assessment and its implementation in software engineering education. Section III describes the experimental

setup: course in study, students background and the study design. The results are reported in section IV. Challenges, limitations, and threats to validity are discussed in Section V. Finally, the conclusion and future work are highlighted in Section VI.

## II. RELATED WORK

For higher education, peer assessment is a useful tool. Several published articles investigated the educational benefits, limitations, challenges, and recommendations for implementing peer assessment in higher education. In software engineering education, particularly, a few research articles studied the implementation of peer assessment, as a few software engineering courses in universities worldwide embed peer-review activities in their projects. A variety of web-based peer assessment tools were used by [6] in Software Engineering Project program at the University of Tasmania to guarantee the integrity of the peer assessment. The authors found that web based peer assessment saves time of the students, provides timely feedback, and helps the lecturer to manage the assessment process. Garousi in 2010 [7] incorporated peer reviews in the design project of a senior software engineering course in two instances of the course in a Canadian university. The author reported his findings, challenges, lessons learned and recommendations. The findings can be used to foster peer-review activities in teaching design projects in other engineering disciplines.

Peer review was employed in [8] by Smith, Tessler, & Kramer in 2012 to teach software testing; peer testing was incorporated into a data structure course at The University of Texas. The authors said that this would allow students to learn from each other, make testing an enjoyable activity, reveal the importance of testing in discovering bugs earlier, and provide a mechanism for evaluating testing skills. They also found that most students enjoyed doing peer testing despite the extra work. In 2020, software engineering students of KU Leuven were engaged in grading for two years [5]. In the first year, “Software Design in C++” course students were asked to do self-evaluation, while the implementation of peer assessment was conducted the next year. The authors of [5]; Groeneveld, Vennekens, and Aerts, explored self-evaluation, motivation, and study time of students in relation to peer assessment, they found that peer assessment could affect positively the reflective and the lifelong learning skills of students. In their study, they primarily focus on peer assessment as a learning tool for students rather than a replacement grading tool for instructors.

Fair assessment of individual contributions to the software engineering group project is not easy. Hemer in 2008 [9] described a method for conducting peer-based assessment of group projects in “Software Engineering and Project course” at the University of Adelaide. Hemer proposed rubrics and developed an online tool to analyze and present data. Hemer found that ad-hoc approaches to peer-assessment is ineffective, rather, a structured, streamlined approach is more recommended. Several ways can be used to assess students in software engineering group projects including assessment by supervisor, Joint Assessment by peers and supervisor, and facilitated peer assessment. Rosen in 2002 [10] explored these ways at the university of Derby and emphasized facilitated peer assessment to extend the learning opportunities through assessment exercise. A quasi-arithmetic approach to group-peer assessment was proposed by Vossen and Ajit in 2020 [11] as an alternative to the conventional arithmetic mean

approach. The proposed approach is relevant for software engineering education where group projects with multiple types of outcomes are used to assess individual students. Two distinct tools for group-peer assessment were developed. The first tool is a simple desktop application using Excel VBA while the second tool is a web-based application. Both tools were used to compare the conventional arithmetic mean and the alternative quasi-arithmetic approaches to prove the superiority of the latter. Vossen and Ajit conducted two pilot studies in two consecutive years on software engineering students at the University of Northampton [11]. In 2022, Ehécatl, Galster, Gilson, and Mathews [4] analyzed the quality and scoring behavior of students in peer evaluation in an undergraduate “Software Engineering Group Project” course over three years in the University of Canterbury. It was found that the quality and scoring of peer evaluations was mostly consistent throughout the course or across students’ calibers. However, they found that qualitative aspects of the peer evaluation were impacted by the caliber of students.

Peer assessment is also crucial in other related engineering disciplines. In the School of Industrial and Computing Engineering in the University of Leo'n, Spain, three courses including Operating systems, Informatics for Industrial Electronics and Informatics for Mechanical Engineering involve assignments require the development of programs written in C language to solve a problem [1]. Conde, Sa'ncchez-gonza'lez, Matella' n-olivera, and Rrodrí'guez-lera in 2017 [1] proposed to assess one assignment using peer review, each student should carry out their assignment and evaluate 3 of their peers. Quantitative and qualitative data were collected and analyzed to evaluate students' opinion and performance. They found that this assessment methodology increased students' participation due to the increase in their motivation. Also, they found that grades were better than in previous editions and that there is a correlation between the students' grades given to their assigned works and teachers' grades for these works. Kern, Pernigotti, Calegaro, and Bento in 2002 [12] introduced an approach to the application of peer review in engineering education to improve written expressions of students of Computer Engineering and Computer Science. They asked students in a “database” course to work as authors of papers and reviewers of their peers. They found that this approach is promising in spite of its difficulty. Numerous software applications exist allowing the teacher to engage students in reviewing each other's products and assessing their contributions in teamwork projects [13]. In [13] de Graaff, Bogaard, and Saunders-Smits developed a peer and self-evaluation instrument for teamwork at TU Delft, which is part of a project sponsored by the Dutch National foundation SURF. They recommend the use of either SPARK or PeEv in project-based learning as practical and easy to use tools.

## III. EXPERIMENTAL SETUP

### A. Course Overview

The course of the study is the “Object-Oriented Software Development” course (OOSDEV); an undergraduate, 3-credit hours, course for second-year software engineering students. It is also an optional requirement for other students in the faculty of Information Technology. This course introduces students to software modelling as a software engineering activity that helps in simplifying complex software systems. The course covers all object-oriented software engineering lifecycle activities and their models, with an emphasis on requirements engineering and design activities. The course

discusses three perspectives of system models (i.e. functional model, dynamic model, and object model). The course uses the popular Unified Modelling Language (UML) as modelling language. Several UML diagrams are introduced in this course including use case diagram, class diagram, sequence diagram, activity diagram, state diagram, and communication diagrams. Furthermore, students are engaged in modelling software systems using UML through case studies and real-life scenarios. During this course, students are required to complete an integrated modelling project assignment presented as a group project, which accounts for 20% of the total score. The course has a single instructor. This article tackles the 2023-2024, fall semester instance of the course, in which 92 students were enrolled.

### *B. Students Background*

All the students enrolled in the “Object-Oriented Software Development” have passed a first-year course “Fundamentals of Software Engineering” which introduces software engineering principles and practices.

### *C. Study Design*

To answer RQ1, a general, customizable rubric was designed by the instructor. The rubric is divided into 6 parts that cover the basic UML diagrams (i.e. use case diagram, sequence diagram, class diagram, activity diagram, state diagram) and use case description. Several criteria were employed to evaluate each part and are aligned with the learning objectives and outcomes of the course. Each criterion has a 5-levels rating scale (excellent, very good, good, fair, poor), with a specific weight given to each criterion left to be determined by the instructor based on their preferences. The general rubric is displayed in Table 1.

To answer RQ2 and RQ3: assessment was carried out in four stages: project assignment preparation, instructor assessment, peer assessment, and data analysis.

#### *1) Project assignment preparation*

Students were distributed into groups of around four students each. The instructor prepared the specifications of the software system to be modelled. The proposed software system was an online submission system for a peer-reviewed journal. Then students were asked to model the software system using basic UML diagrams; in specific, use case diagram, sequence diagram for one of the use cases, class diagram, activity diagram, a state diagram for a specific object, and a detailed description of a specific use case. The project assignment was published using an online platform (MS. Teams) and students were given one month to complete and submit their group work. Meanwhile, the instructor customized the proposed general rubric, displayed in Table 2, to produce a version that better fits the current-assignment context and gave a weight for each criterion. This version was used in both instructor assessment and peer assessment to guarantee internal consistency, and guide students through the assessment process. Alongside, the instructor prepared a solution model key to be used later to compare solutions in the instructor assessment and the peer assessment phases.

#### *2) Instructor assessment*

Once students have submitted their work, the instructor assessed the work for each group according to a pre-designed solution key model and the customized rubric in table 2. Then, a score of 20 was calculated.

#### *3) Peer assessment*

Once the instructor assessment stage had completed, peer assessment started. Solution model key beside the customized rubric were published. Anonymous projects were distributed among groups in peers, and groups were asked to do the peer assessment. The allocation of reviewers to reviewees was done randomly in pairs. In this research, a one-to-one persistent group reviews strategy was used. Therefore, each project group had to evaluate one other project, assign a score for each criterion based on the given weights, calculate the total score, and provide the instructor with their assessment through the online platform (i.e. MS Teams), only one score is required by the group agreed by all members. Also, students were given space in the evaluation form to write their feedback on their peer’s work. In order to encourage students, do the assessment, the assessment was considered part of the project and students were promised to be rewarded with extra points for those who do the assessment reasonably. Students were given enough time to do the assessment.

#### *4) Data analysis*

To answer RQ2, we investigate the degree of agreement between peer assessment and the instructor or expert assessment by checking the validity of the peer assessment. First, the scores were extracted from the assessment spreadsheets; for each project, instructor score and peer score were assigned. Then, correlation analysis between the instructor scores and the students peer score was conducted and Pearson correlation coefficient was calculated.

To answer RQ3, we measured the deviation of the peers’ assessment from the instructor’s assessment, then conducted a correlation analysis between the standard deviations and the instructor assessment of the evaluators.

## IV. RESULTS AND DISCUSSION

The peer assessment alongside the instructor assessment for each project is displayed in Table 2. Fig. 1 compares the instructor assessment and the student peer assessment for each project. The chart reveals a good level of agreement among the instructor and students peer scores for each project. The collected data was analyzed to help us estimate the validity of the peer assessment. To assess the accuracy of peer assessment, correlation between peer and instructor assessments was investigated. The Pearson correlation coefficient was calculated with a value of ( $r=0.71$ ,  $p=0.001$ ) which indicates a strong, positive, significant correlation, which implies a good agreement between the instructor and the peer assessment. This means that students in the “OOSDEV” course evaluate each other often properly.

The instructor assessment for the evaluator of each project is assigned in Table 2. Moreover, the deviation of the peer assessments from the instructor assessments is also displayed in the same table. Then, we conducted a correlation analysis between the standard deviations and the instructor assessments of the evaluators. We obtained a Pearson correlation coefficient of ( $r=-0.57$ ,  $p=0.009$ ) which indicates a strong, negative, significant correlation. This correlation is displayed in Fig. 2. This result means that good students’ groups (i.e. who got good instructor assessment) assess their peers more properly as they have less deviation from the instructor assessment.

TABLE I. BASIC UML DIAGRAMS ASSESSMENT RUBRIC

Criteria	level	Excellent (All) (81- 100)%	Very Good (most) (61-80)%	Good (Many) (41-60)%	Fair (Some) (21-40)%	Poor (few or no) (0-20)%
<b>Use case diagram ( pts)</b>	<b>Weight</b>	( ) pts	( ) pts	( ) pts	( ) pts	( ) pts
1) Actors are identified						
2) Relationships between actors are well specified						
3) Use cases are properly identified						
4) <>extend>> relationships are properly assigned						
5) <>include>> relationships are properly assigned						
<b>Score: ( out of ), ( out of )</b>						
<b>Sequence diagram ( pts)</b>	<b>Weight</b>	( ) pts	( ) pts	( ) pts	( ) pts	( ) pts
1) Participating actors are identified						
2) Participating objects identified; their roles (boundary, control, entity) objects are clearly used						
4) Messages are properly specified						
5) Sequence of messages is properly described						
6) Combined fragments (alt and loops) are properly used						
7) Objects creation time is considered in the diagram						
<b>Score: ( out of ), ( out of )</b>						
<b>Use case description ( pts)</b>	<b>Weight</b>	( ) pts	( ) pts	( ) pts	( ) pts	( ) pts
1) Participating actors are identified						
2) Pre-conditions (entry) are well described						
3) Post-conditions (exit) are well described						
4) Basic flow steps are primitive, well described and sequentially numbered						
5) Other existing use cases are included in the basic flow correctly						
6) Alternative and exceptional use cases are specified with extension points in the alternative flow						
7) If applicable, nonfunctional requirements and exceptional use cases with no specific extension points are described						
8) use case name is clearly identified						
9) Use case description matches the use case diagram						
10) No redundancy in the use case description sections						
<b>Score: ( out of ), ( out of )</b>						
<b>Class diagram ( pts)</b>	<b>Weight</b>	( ) pts	( ) pts	( ) pts	( ) pts	( ) pts
1) Classes are identified						
2) Data fields are specified						
3) Operations and their parameters are specified						
4) Inheritance relationships are specified						
5) Associations are specified with meaningful names						
6) Required references are specified						
7) Obvious multiplicities are specified						
8) Required Enumerations are specified and properly associated to classes						
<b>Score: ( out of ), ( out of )</b>						
<b>Activity diagram ( pts)</b>	<b>Weight</b>	( ) pts	( ) pts	( ) pts	( ) pts	( ) pts
1) Activities are identified						
2) Swimlanes are identified, and activities are assigned to them properly						
3) Sequencing among activities is well described						
4) Alternatives, branch and merge nodes, guard expressions, are used properly						
5) Synchronization, fork and join nodes are used properly						
<b>Score: ( out of ), ( out of )</b>						
<b>State diagram ( pts)</b>	<b>Weight</b>	( ) pts	( ) pts	( ) pts	( ) pts	( ) pts
1) States are identified properly						
2) Transitions are described properly						
3) Initial and final nodes are stated correctly						
4) Identified states are for the same single object						
<b>Score: ( out of ), ( out of )</b>						
<b>Total Score: ( out of )</b>						

TABLE II. CUSTOMIZED UML ASSESEMENT RUBRIC FOR THE CURRENT INSTANCE OF THE SOFTWARE PROJECT

Criteria	level	Excellent (All) (81- 100)%	Very Good (most) (61-80)%	Good (Many) (41-60)%	Fair (Some) (21-40)%	Poor (few or no) (0-20)%
<b>Use case diagram (4 pts)</b>	<b>Weight</b>	<b>(4) pts</b>	<b>(3) pts</b>	<b>(2) pts</b>	<b>(1) pts</b>	<b>(0) pts</b>
1) Actors are identified	5					
2) Relationships between actors are well specified	4					
3) Use cases are properly identified	33					
4) <>extend>> relationships are properly assigned	8					
5) <>include>> relationships are properly assigned	10					
<b>Score: ( ___ out of 20) , ( ___ out of 4)</b>						
<b>Sequence diagram (3 pts)</b>	<b>Weight</b>	<b>(2) pts</b>	<b>(1.5) pts</b>	<b>(1) pts</b>	<b>(0.5) pts</b>	<b>(0) pts</b>
1) Participating actors are identified	2					
2) Participating objects identified; their roles (boundary, control, entity) objects are clearly used	4					
4) Messages are properly specified	13					
5) Sequence of messages is properly described	2					
6) Combined fragments (alt and loops) are properly used	2					
7) Objects creation time is considered in the diagram	2					
<b>Score: ( ___ out of 12) , ( ___ out of 3)</b>						
<b>Use case description (3 pts)</b>	<b>Weight</b>	<b>(2) pts</b>	<b>(1.5) pts</b>	<b>(1) pts</b>	<b>(0.5) pts</b>	<b>(0) pts</b>
1) Participating actors are identified	2					
2) Pre-conditions (entry) are well described	2					
3) Post-conditions (exit) are well described	2					
4) Basic flow steps are primitive, well described and sequentially numbered	4					
5) Other existing use cases are included in the basic flow correctly	2					
6) Alternative and exceptional use cases are specified with extension points in the alternative flow	4					
7) If applicable, nonfunctional requirements and exceptional use cases with no specific extension points are described	NA					
8) use case name is clearly identified	2					
9) Use case description matches the use case diagram	2					
10) No redundancy in the use case description sections	2					
<b>Score: ( ___ out of 18) , ( ___ out of 3)</b>						
<b>Class diagram (4 pts)</b>	<b>Weight</b>	<b>(4) pts</b>	<b>(3) pts</b>	<b>(2) pts</b>	<b>(1) pts</b>	<b>(0) pts</b>
1) Classes are identified	11					
2) Data fields are specified	17					
3) Operations and their parameters are specified	24					
4) Inheritance relationships are specified	4					
5) Associations are specified with meaningful names	16					
6) Required references are specified	24					
7) Obvious multiplicities are specified	14					
8) Required Enumerations are specified and properly associated to classes	4					
<b>Score: ( ___ out of 32) , ( ___ out of 4)</b>						
<b>Activity diagram (4 pts)</b>	<b>Weight</b>	<b>(4) pts</b>	<b>(3) pts</b>	<b>(2) pts</b>	<b>(1) pts</b>	<b>(0) pts</b>
1) Activities are identified	24					
2) Swimlanes are identified, and activities are assigned to them properly	4					
3) Sequencing among activities is well described	4					
4) Alternatives, branch and merge nodes, guard expressions, are used properly	5					
5) Synchronization, fork and join nodes are used properly	NA					
<b>Score: ( ___ out of 20) , ( ___ out of 4)</b>						
<b>State diagram (2 pts)</b>	<b>Weight</b>	<b>(2) pts</b>	<b>(1.5) pts</b>	<b>(1) pts</b>	<b>(0.5) pts</b>	<b>(0) pts</b>
1) States are identified properly	6					
2) Transitions are described properly	7					
3) Initial and final nodes are stated correctly	2					
4) Identified states are for the same single object	2					
<b>Score: ( ___ out of 8) , ( ___ out of 2)</b>						
<b>Total Score: ( ___ out of 20)</b>						

TABLE III. ASSESSMENT RESULTS

Proj. ID	Instructor Assessment	Peer Assessment	SD	Instructor Assessment (of evaluator)
proj1	10.39	6.70	2.609	7.88
proj2	7.21	9.00	1.266	7.58
proj3	8.93	10.50	1.110	12.1
proj4	7.58	6.50	0.764	12.9
proj5	15.75	14.60	0.813	7.31
proj6	12.84	11.00	1.301	7.88
proj7	15.05	15.19	0.100	10.75
proj8	8.89	11.60	1.916	10.36
proj9	2.61	4.90	1.619	13.62
proj10	14.11	14.21	0.071	12.33
proj11	7.88	5.00	2.036	10.39
proj12	7.58	9.75	1.534	7.21
proj13	12.1	12.00	0.071	8.93
proj14	12.9	6.10	4.808	7.58
proj15	7.31	7.60	0.205	15.75
proj16	7.88	9.00	0.792	12.84
proj17	10.75	10.20	0.389	15.05
proj18	10.36	7.10	2.305	8.89
proj19	13.62	18.70	3.592	2.61
proj20	12.33	10.18	1.524	14.11

## V. CHALLENGES, LIMITATIONS, AND THREATS TO VALIDITY

Peer assessment suffers from several challenges. One challenge is that students normally do not trust peer assessment. Random grading was avoided by motivating students to evaluate properly. Also, students think of peer assessment as extra workload and kind of waste of time. Hence, we used a one-to-one peer assessment in order to not to over burden students. As well, the instructor of the course also suffered from work overload as she spent extra time and effort setting up the current experiment with a desire to probably lessen instructors' workload in the future. Another challenge was that students were not familiar with the assessment process besides the lack of time in the beginning of the course to train students on how to do the peer assessment. A short guidance video was delivered to students to assist them preceding the peer assessment stage. One limitation of the study is the sample size. The study was conducted on 20 projects delivered in one semester: fall 2023-2024. This leads to another limitation which is the longitude of the research. The following threats to validity were identified:

External Validity: as the study was conducted for a specific instance of a course in one university, the findings cannot be generalized to other courses or students without careful analysis.

**Internal Validity:** Social relations among students can be a threat to validity, to mitigate from this, projects were distributed in peers anonymously. In this instance of the course, no selection bias was present as all the students enrolled in the course were invited to participate (in groups); only two groups lagged, they were reminded before they sent their assessment. However, members of the project groups were chosen by the students themselves. The same distribution was used in peer assessment groups. Other studies need to be conducted to ensure whether this would affect the internal validity. It is strongly believed that teams vary in their standards and strength, and this would probably affect results. The result of studying the correlation between the deviation of the peer assessment from instructor assessment and the instructor assessment of the evaluators would advocate this. In other words, we can say that weaker students may be less critical and accurate than stronger students. Moreover, the same tool (MS. Teams) was used for collecting the peer evaluations which mitigates the instrumentation threats.

**Construct Validity:** A detailed rubric for assessing UML projects was designed, shared, and used by all peer groups and the instructor in evaluating the project assignment. Instructor's scores were given to students before they had done the peer assessment. However, to preserve construct validity, instructor's scores were given to students individually throughout their individual accounts on their portals without any project identifier. Only the highest score was made public, which might slightly affect the results.

**Conclusion Validity:** The study was also based on the assessment of a single instructor; this may lower the validity and reliability as expert ratings sometimes exhibit inconsistency [14]. Also, the study was based on one-to-one peer raters. However, using additional raters and considering the mean score would improve reliability and thus conclusion validity.

## VI. CONCLUSION AND FUTURE WORK:

This work contributes to investigate the validity of peer assessment of students in a modelling project assignment in the Object-Oriented software development course at the department of Software Engineering, the Hashemite University. The article presented the author's experimental findings, lessons learned, possible challenges, and recommendations from applying peer assessment into project-based software engineering courses. The findings were promising and can be used to encourage other instructors to embed peer assessment activities in teaching other courses, even in other disciplines.

However, major shortcomings could be mitigated by conducting the study on several semesters. This is one direction in the future to increase the sample size and obtain more convenient results. Another direction is to add several raters (both experts and students) to improve the reliability of peer assessment. Also, we intend to design and deliver a web-based peer assessment tool that helps in managing the peer assessment process. From another point of view, further studies could be conducted in the future to investigate whether students peer assessment can be an adequate substitute for instructor assessment.

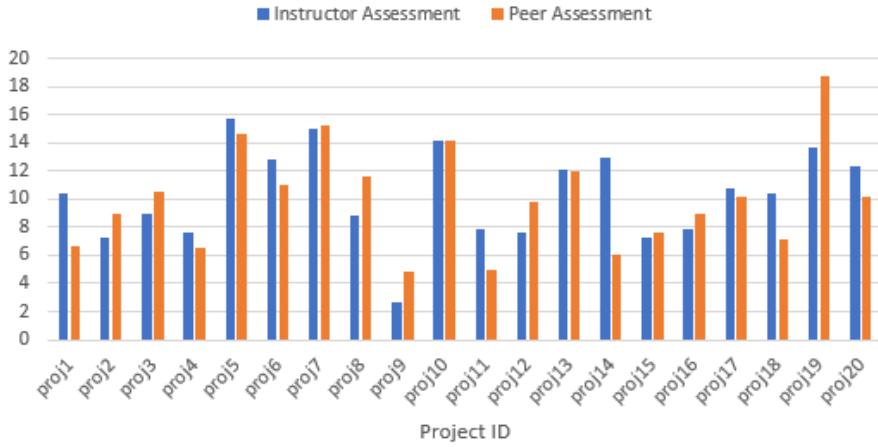


Fig. 1. Instructor vs. Peer Assessment

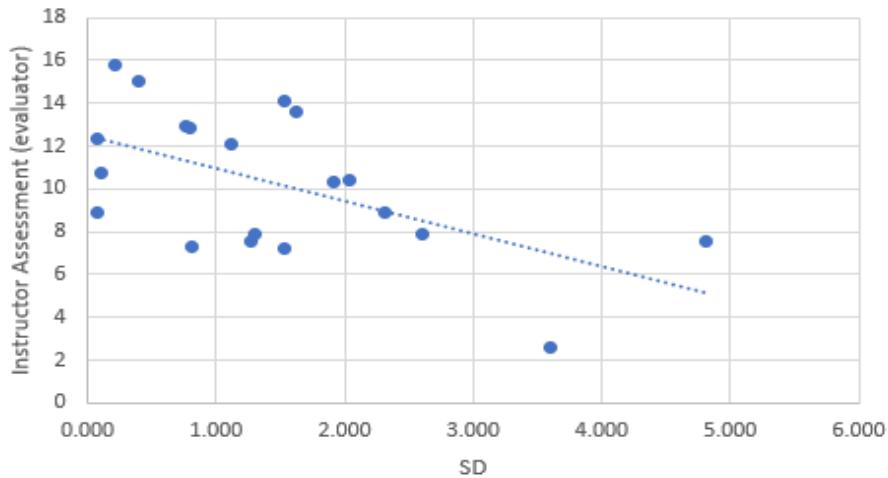


Fig. 2. Instructor Assessment of Evaluators vs. Standard Deviation

## REFERENCES

- [1] M. A. CONDE, L. SA' NCHEZ-GONZA' LEZ, V. MATELLA' N-OLIVERA and F. J. RODRI'GUEZ-LERA, "Application of Peer Review Techniques in Engineering Education," International Journal of Engineering Education, vol. 33, no. 2, pp. 918–926., 2017.
- [2] G. V. HELDEN, V. V. DER WERF, G. N. SAUNDERS-SMITS and M. M. SPECHT, "The Use of Digital Peer Assessment in Higher Education – an Umbrella Review of Literature," IEEE Access, vol. 11, pp. 22948 - 22960, 2023.
- [3] C. Adachi, J. Hong-Meng Tai and P. Dawson, "Academics' perceptions of the benefits and challenges of self and peer assessment in higher education," Assessment & Evaluation in Higher Education, pp. 294-306, 2017.
- [4] M. M.-T. Ehécatl , M. Galster, F. Gilson and M. Mathews, "A Three-Year Study on Peer Evaluation in a Software Engineering Project Course," IEEE TRANSACTIONS ON EDUCATION, vol. 65, no. 3, pp. 409 - 418, 2022.
- [5] W. Groeneveld, J. Vennekens and K. Aerts, "Engaging Software Engineering Students in Grading: The effects of peer assessment on self-evaluation, motivation, and study time," arXiv.org 7, 2020.
- [6] N. Clark, P. Davies and R. Skeers, "Self and Peer Assessment in Software Engineering Projects," in Proceedings of the 7th Australasian conference on Computing education, 2005.
- [7] V. Garousi, "Applying Peer Reviews in Software Engineering Education: An Experiment and Lessons Learned," IEEE TRANSACTIONS ON EDUCATION,, vol. 53, no. 2, pp. 182-193, 2010.
- [8] J. Smith, J. Tessler and E. Kramer, "Using peer review to teach software testing," in ICER '12: Proceedings of the ninth annual international conference on International computing education research, 2012.
- [9] D. Hemer, "Peer Assessment of Group-based Software Engineering Projects," in 19th Australian Conference on Software Engineering, 2008.
- [10] C. C. Rosen, "Individual assessment of group projects in software engineering: a facilitated peer assessment approach," in Proceedings of 9th Conference on Software Engineering Education, 2002.
- [11] P. H. Vossen and S. Ajit, "Towards a Paradigm Change in Group and Peer Assessment in Software Engineering Education," in Training, 32nd IEEE Intl. Conference on Software Engineering Education &, 2020.
- [12] V. M. Kern, J. M. Pernigotti, M. M. Calegaro and M. Bento, "Peer review in engineering education: speeding up learning, looking for a paradigm shift," in Seventh International Conference on Engineering and Technology Education, 2002.
- [13] E. D. de Graaff, M. E. v. d. Bogaard and G. N. Saunders-Smits, "PEER Assessment and PEER review to support teamwork in engineering education," in 6th ASEE Global Colloquium on Engineering Education, Istanbul, 2007.
- [14] M. WARD, L. GRUPPEN and G. REGEH, "Measuring Self-assessment: Current State of the Art," Advances in Health Sciences Education, vol. 7, p. 63–80, 2002.
- [15] N. Dahal, B. C. Luitel, B. P. Pant and R. Rajbanshi, "ENHANCING STUDENT-TEACHERS ASSESSMENT SKILLS: A SELF-AND PEER-ASSESSMENT TOOL IN HIGHER EDUCATION," International Journal of Education and Practice, vol. 10, no. 4, pp. 313-321, 2022.



# Case Study Based Pedagogical Intervention for Teaching Software Engineering Ethics

Aditya Johri  
Information Sciences & Technology  
George Mason University  
Fairfax, VA USA  
[johri@gmu.edu](mailto:johri@gmu.edu)

Ashish Hingle  
Information Sciences & Technology  
George Mason University  
Fairfax, VA USA  
[ahingle2@gmu.edu](mailto:ahingle2@gmu.edu)

**Abstract**—The omnipresence of software systems across all aspects of society has necessitated that future technology professionals are aware of ethical concerns raised by the design and development of software and are trained to minimize harm by undertaking responsible engineering. This need has become even more urgent with artificial intelligence (AI) driven software deployment. In this paper we present a study of an interactive pedagogical intervention – role-play case studies – designed to teach undergraduate technology students about ethics with a focus on software systems. Drawing on the *situated learning* perspective from the Learning Sciences, we created case studies, associated stakeholder roles, discussion scripts, and pre and post discussion assignments to guide students' learning. Open-ended data was collected from thirty-nine students and analyzed qualitatively. Findings from the study show that by taking on different perspectives on a problem, students were able to identify a range of ethical issues and understand the role of the software system process holistically, taking context, complexity, and trade-offs into account. In their discussion and reflections, students deliberated the role of software in society and the role of humans in automation. The curricula, including case studies, are publicly available for implementation.

**Keywords**—software engineering ethics, ethics education, case studies, role plays, situated learning

## I. INTRODUCTION

Whether it is self-driving cars, recommendation systems for shopping, or applying for a loan, the impact of software on people and society is now inescapable [1-2]. As software gets increasingly embedded across technological artifacts in society, the need to better understand its intended, unintended, anticipated, and unanticipated impacts becomes more acute. The increased use of complex algorithms, machine learning, and the advent of artificial intelligence (AI), which are harder to understand but even more impactful, intensify the need for ethical and responsible software development [3-4]. The field of software engineering ethics, which examines these issues, has thus become increasingly critical within software engineering (SE). Yet, within software engineering education, the topic of ethical responsible design, development, and deployment is neglected and needs to be paid more attention. Given the importance of ethics to software engineering, other scholars have also advocated for a stronger position within SE on ethical education [5] that “takes into account both rules and consequences to assess the goodness of actions, and at the same

time pays an adequate consideration to the absolute values of human dignity” [6, pg. 505].

Over the years, software engineering ethics has primarily focused on two aspects of the process: the choices made by engineers or developers and the professional codes that guide their practices and software development. A focus on professionals in the field is essential, as they design and develop the systems that are implemented across industries and organizations. Since the user facing software engineering work is largely done in the industry, unsurprisingly, the focus on ethics has been on professionals. We argue that it is equally important to focus on students or future software engineers and their training related to software engineering ethics. If students are exposed to ethical concerns early on, they can learn to think holistically about responsible development and use of software. Higher education is also a time in their education and career when they have time and support to delve more into ethics.

Many scholars have recently developed ways to teach ethics to provide stronger contextual training to students [7-8]. In this paper, we present a research study of a pedagogical intervention designed to teach students the ethics of software in an interactive manner through the use of role-play case studies. Role-play case studies are a form of simulation exercise or training that provides students with a real or fictional scenario and narrative, assigns them roles, and asks them to deliberate and discuss a possible solution [9]. The ability to make ethical decisions relies to a large extent on the ability of the decision-maker to take context into account to understand not just the immediate technical need of the work but also larger implications that might even result from unanticipated consequences. We designed multiple case studies related to the design and use of software that were conducive to role-play. We implemented them in a class for information technology students that focused on teaching the social and ethical implications of technology. We collected data using multiple sources – survey, focus group, discussion transcript – and analyzed them iteratively to understand student learning outcomes. Overall, we found that our intervention and approach led to a measurable change in students' understanding of the ethics of software.

In the rest of the paper, we first review prior work on software engineering ethics, situated learning – the theoretical perspective underpinning our approach, and the use of case studies in ethics education. This is followed by a discussion of the design of role-play case studies and the context of

implementation, including course and student details. Research study and findings follow this, and we end with a discussion and conclusion section. Our intervention has so far been implemented across ten class sections over four years (see [43-45] for details), but for the purposes of this paper, we limit the research study to multiple cases over one course section to provide a comprehensive and in-depth assessment of learning outcomes.

## II. PRIOR WORK

### A. Software Engineering Ethics

Software engineering ethics is a multifaceted field that examines several interrelated approaches to the ethical development of software. Gotterbarn in particular, has made significant contributions to the field starting with the software code of ethics advanced by ACM in 1997 [10-12]. It emphasizes understanding the practical decision-making of software engineers as well as the creation of guidelines or codes that guide software development [13-15]. The field is founded on the principle that technical decisions and human values are intertwined and software engineers have can make a “significant positive impact on society, simply by becoming more sensitive to the long-term human relations implications of our work and incorporating this sensitivity into our software designs and products” [16].

Two recent works within the field are of particular relevance to the pedagogical approach and subsequent assessment reported here. First, an applicable framework that has been advanced is the Ethically-Driven Software Design (EDSD) [17]. EDSD is a practical approach to software development that creates a link between ethical and professional skills making the overall process more transparent and also brings in an awareness of risks and limitations involved. The application of this framework is through the use of specific index cards to aid the design of software [17]. These cards prompt responding to an ethical question from the viewpoint of different stakeholders involved with the software design process including customer, manager, engineer, etc. The design and use of these cards aligns with the role-play-based intervention we have designed. In both cases, the ability to take a perspective and offer a viewpoint is central to the design or learning process. The need to use or play a role ensures that multiple stakeholder values are represented. This ability is crucial to both documenting and addressing ethical concerns.

Second, in their roadmap for ethics-aware software engineering, Aydemir and Dalpiaz [18] propose an analytical framework that assists stakeholders in analyzing ethical issues in terms of subject (software artifact or SE process), relevant value (diversity, privacy, autonomy, etc.), and threatened object or stakeholder (user, developer, etc.). More critically, from the perspective of the current study, they identify a range of ethical concerns that are relevant for pedagogy including transparency, awareness, privacy, accountability, diversity, dependability, and the role of business and work ethics within the software development process. Overall, the elements they identify align closely with other similar ethical and responsible software development aspects that have been advanced, especially in relation to algorithmic and AI-driven systems. For the purposes of this study, their list provides a mechanism to analyze the data

to identify ethical aspects that students bring up in their discussions and responses.

Overall, prior work on software engineering ethics emphasizes examining software engineering and the implications of software use from a more contextual perspective considering viewpoints of different stakeholders. Similar to other ethical frameworks, it reiterates taking responsibility for developing software and doing no harm [19-21].

### B. Situated Learning Perspective

One of the theories of learning that allows us to design and assess pedagogy that focuses on teaching paying attention to contextual aspects of an issue is situated learning [22-24]. The situative perspective emphasizes the role of context on scaffolding an individual’s conception of knowing and how they learn; according to situative learning, knowledge is not something that an individual possesses or stores in the brain but is present in all that they do [25]. Consequently, people learn best through working on socially relevant problems in collaboration with others.

The idea within the situative perspective that is particularly relevant to the proposed work is that what we learn and how we learn it is closely linked [26]. This situation, or context, determines what people look for, how they get the information, and how they interpret it. Knowledge arises dynamically and is constantly constructed and reinterpreted. Therefore, both in terms of how to teach and what to teach, this viewpoint highlights the importance of context and social practices. Within the situated learning paradigm, Greeno and van de Sande [27] further advance a perspectival view which argues that a person’s or group’s knowledge and understanding of any conception is “their ability to construct perspectival understandings that are situated in activity and that are organized according to principles that are taken as defining the conception (pg. 14)”. This motivated our use of case studies and, specifically, role-plays as pedagogical tools that can support ethical thinking among students, and place ethics more centrally within the course and curriculum [28].

### C. Case Based Ethics Education

Ethics education has a long history within higher education [29]. Case studies are short narratives that condense a real-world problem or concern and can be focused on a narrow problem or a larger-scale project [30-31]. They are commonly used for teaching ethics within engineering and computing and employed both to teach how to apply professional codes or even broader societal considerations of engineering. The use of case studies for education originated in business schools and continues to be a popular form of teaching in that field. [30] have proposed that case studies are also good for teaching software engineering. Case studies and, in particular, role-play-based discussions are effective because, as prior research on situated learning has demonstrated, there is value in inculcating perspectival thinking and that case studies and collaborative discussions are effective in achieving this learning outcome.

Role-playing, a specific type of case discussion informed by simulation-based pedagogy, involves using cases or scenarios to discuss a problem or an issue where each participant is assigned a specific role [9, 33, 34]. Within the context of teaching,

students take on the role of a person who is involved with or impacted by the problem outlined in the case, and they have to respond to questions and take part in a discussion in a manner that reflects the viewpoint or perspective of the role they are assigned. Role-play supports learning across a range of areas, including perspective-taking, critical thinking, and communication skills [35-36]. Furthermore, a role-play-based approach lends itself to a broader discussion around the role of technology from multiple perspectives.

Within software development and software engineering, in addition to the actual algorithms, programming environments, and organizational details, there is the need to better understand the use of a specific software system at the point of implementation. The usefulness and subsequent continued use depend on how users are able to integrate a system in their work and life practices. This also implies that to reduce any unanticipated outcomes that can be harmful, it is important for an engineer to take the overall context into account. This is where a focus on ethics is important, and the use of pedagogical strategies that teach students how to build a mindset that supports their intention for ethical and responsible development. Within software engineering pedagogical practices such as role-based group work, scenarios, ethical deliberation, and social engaged learning are already being used in some form, and our approach is an adaptation and extension of this work [37-40].

### III. CASE DEVELOPMENT AND IMPLEMENTATION CONTEXT

We leveraged prior work on the design of role-play case studies to create four software-related cases. Each case included a real-life inspired scenario encapsulating a software-driven problem within a larger context.

*1) Boeing Case:* This case focuses on the Boeing 737 Max aircraft crashes, specifically, the use and implementation of the Maneuvering Characteristics Augmentation System or MCAS software and Boeing's failure to inform the pilot about MCAS.

*2) Volkswagen Emissions Case:* This case focuses on the use of a software system by Volkswagen to cheat on emissions testing for its diesel engine vehicles in the U.S. and the aftermath of that scandal.

*3) Facial Recognition Case:* The case focuses on the use of facial recognition by a university to track people on campus to ensure they are fully compliant with post-Covid requirements.

*4) Credit and Lending Case:* This case focuses on the use of novel forms of data and models by a bank to make credit and lending decisions.

For each case, we created at least six to eight different roles, including a moderator role - a stakeholder who would lead the discussion and engage all the other participants. Each case also included relevant readings and video resources. Some resources focused on specific use cases of technology, while others were broader and introduced ethical frameworks or ideas. Students were encouraged to undertake additional research to prepare for their role, and most reported doing so. Table 1 summarizes the case, participant roles, and ethical elements the case targeted. *Additional information about the cases, including the full text, is available online publicly and will be added post-review.*

We implemented the cases across ten class sections with 30-45 students each between 2020 – 2024. The students were divided into groups of 5-7 for the role-play discussions, and these groups remained largely consistent. In each course offering, at least four cases were implemented. We used a design-based research approach to understand the implementation outcome and evaluate the intervention (anderson2012design}). Based on student feedback and our observations, we systematically revised aspects of the cases and their implementation in different iterations. For instance, we added more background reading about technical concepts across all the cases, created more roles for larger sections, and recorded a mock video demonstrating how to run a role-play.

TABLE I. CASE DETAILS

Case	Description	Roles	Ethical Software Concerns in the Case
Boeing	As a member of the Aviation Transportation Investigative Committee (ATIC), participants are tasked with discussing how systematic (technical and social) issues in aviation can be mitigated.	Aviation Consultant Aerospace Eng. Professor Software Eng. Retired FAA Officer Boeing Representative Family Lawyer	Accountability Transparency Business concerns Dependability Work concerns Safety
Volkswagen	A rental company's fleet manager is interested in VW vehicles. They want to know if the company can be trusted and what responsibility organizations have to correct the problems they cause.	Consumer Adv. Group Chair Env. Engineering Professor Env. Advisory Group Director Software Developer Automotive Industry Expert Compliance Director	Trustworthiness Sustainability Business concerns Work concerns Accountability
Facial Recognition	In response to the global pandemic, Andrew Hamilton University administrators have established a task force to discuss if facial recognition technology will ensure students' safe return on campus.	V.P. of Campus IT V.P. of Student Government History Professor A.V.P. in Provost's Office Equity and Inclusion Director FRT Non-Profit Director	Privacy Surveillance Diversity Transparency Accountability
Credit and Lending	A loan manager wants to explore algorithmic decision-making for credit and loan decision-making and whether data on demographic and socio-economic profiles should be included.	Data Analyst Underwriting Specialist Chief Credit Innovation Officer Financial Research Analyst Loan Process Regulatory Officer AI Ethics Adv. Group Director	Accountability Transparency Bias Diversity Privacy

### B. Course Details

The course, consistent across the four years of this work, was an undergraduate offering in IT designed to teach students about the role of technology in society. The course fulfills the ABET accreditation related to ethical and professional responsibilities for students in degree programs, including cybersecurity, information technology, and data analytics (broadly described as technology students). The students in the program take programming and software development as a required course. They also undertake software development in their capstone course either concurrently or after this course. Most students graduate to enter the IT industry as programmers/software developers.

The course itself was divided into 2 - 3 week long modules. In the first week, students were introduced to a topic through papers and videos and provided the case study and assigned roles. In the second week, students were given pre-discussion assignments, which included answering a set of questions from the perspective of their role and drawing a concept map on their recognition of broader concepts. In the third week, students would participate in the role-play discussion activity during the class session. In some cases, weeks 2 and 3 were combined. After the session, students would complete a post-role-play assignment and a collaborative group concept map. In some instances, they held a debrief discussion rather than drawing a concept map. Table II lists the steps for role-play application.

TABLE II. APPLICATION OF ROLE-PLAY IN CLASS

1	Introduction to module (topic)
2	Videos covering module material
3	Readings covering module(includes news & scholarsly articles)
4	Distribute the role-play scenarios and assign roles
5	Pre-discussion questions
6	Role-play activity in class
7	Post-discussion questions
8	Debrief and discussion

### IV. RESEARCH STUDY

The overall study was designed as a mixed-methods approach with a mix of data. We created and used different assessments for student learning, including open-ended questions given to students through pre-and-post assignments and concept maps to map the expansion of student concept recognition. We also used focus groups and end-of-semester evaluation data to assess the efficacy of the role-lays. This research was approved by the Institutional Review Board (IRB), and only data from students who consented to the study was analyzed. To protect the confidentiality of participants, data analysis took place after the final grades were assigned, and all data was anonymized by a third party not involved with grading.

Forty-one (41) undergraduate students participated in the course, of which thirty-nine (39) consented to the study and engaged with the role-play case studies. The students' degree programs varied, but all students were required to take at least one software development course before or concurrently with this course where data was collected.

### A. Data Collection

For this study, the data used for analysis included pre-discussion assignments (completed before the role-play activity), post-discussion assignments (completed after the role-play activity), and end of the semester assignments.

The pre-discussion assignment included the following questions: What role are you playing in the role-play group discussion? From the perspective of your role, what will be your recommendation? From the perspective of your role, how can we ensure future safety and transparency and rebuild trust? Why do you think the approach you suggest is the best approach? What do you think are the main barriers to this approach? The post-discussion assignment asked the following questions: What response was reached following the discussion and what criteria were considered to reach this solution? Was the response agreed to by all or did one person have more influence? Why? Do you personally agree with the response? Why/Why not? Any comments on how your group approached the case? There were minor variations in these questions based on the specific case.

The final assignment asked them what they learned about ethics from the course and also individually from each case study. Finally, the primary author moderated the role-play sessions and was cognizant of the discussions that took place. Data was curated by the second author.

### B. Data Analysis

Analysis was done iteratively by both authors. Since in this particular study we were interested in understanding student learning using open-ended responses, a qualitative research approach was deemed appropriate [41]. Consistent with qualitative data analysis, data were open-coded, and then specific items, such as ethical principles and connections to the cases were identified in the data [42]. Once the authors were familiar with the data, they read through and tagged the student responses for common ideas consistently brought up in the discussion. Table III highlights these common ideas as "initial codes", specifically how often the aspects of ethical and responsible software were discussed through the dataset. These initial codes were explored in more detail through the findings. Both authors did multiple round of coding to ensure consistency and reliability, and to select examples for this paper.

### V. FINDINGS

The purpose of the course and the case studies was to introduce students to different aspects of ethical and responsible software engineering. Consequently, cases touched upon different aspects of ethics with some overlap (see Tables I & III). In this section, we present findings grouped first by the different aspects of ethics and then from a more situated perspective to capture students' understanding of the complexity of the

engineering process. To support and illustrate examples, quotes from the students are included. The quotes were minimally edited for anonymity and editing, but care was given not to alter meaning. Student numbers in [square brackets] used with the quotes refer to their corresponding deidentified number in our dataset.

TABLE III. ETHICAL CONCERN Instances

<i>Initial Codes</i>	<i>Boeing</i>	<i>Volkswagen</i>	<i>Facial Recognition</i>	<i>Credit and Lending</i>
<b>Accountability</b>	7	5	4	6
<b>Trust /Dependability</b>	6	<b>8</b>	3	2
<b>Bias</b>	0	1	7	<b>22</b>
<b>Diversity</b>	0	0	<b>10</b>	1
<b>Privacy</b>	0	0	<b>20</b>	4
<b>Safety</b>	8	3	<b>10</b>	0
<b>Sustainability</b>	0	<b>18</b>	1	1
<b>Surveillance</b>	0	1	<b>25</b>	0
<b>Transparency</b>	5	6	3	<b>11</b>
<b>Business Concerns</b>	<b>15</b>	6	0	9
<b>Work Concerns</b>	<b>17</b>	4	1	5

#### A. Aspects of Ethical and Responsible Software Discussed

a) **Accountability:** Different stakeholders are involved in any system development or software engineering project. Therefore, the accountability in the case of an error or fault is an important element to identify. Through the case studies, students were able to discuss the issue of accountability and through the Volkswagen case study, they learned that many companies do take accountability and make amends for the hards they have caused.

*"Volkswagen's response to the scandal [was] to blame the software engineers, or 'rogue' software engineers. The blamed software engineers solely for programming the ECU to make the engine go into a weaker mode... as if the software engineers would care at all to implement it in the first place. As if Volkswagen had no testing from the hardware aspect, or no actual QA or testing of the cars. It took years for this to be discovered... where eleven million cars rolled out of the factory like this, and they completely push the blame off themselves and just blame it on the software engineers who worked there - as if they get their kick out of reprogramming these cars for company profit that they'll never see." [Student 12]*

*"Boeing was not as transparent with their problems as they could have been. It seemed that they tried to blame other factors outside of their plane/product before admitting there was a problem." [Student 1]*

b) **Trustworthiness/Dependability:** Dependability refers to the degree to which an organization or a person can be counted on to fulfill their responsibilities, including ensuring that systems work as intended and that users are fully aware of the outcomes of their actions while using a system. Dependability, in some ways, is related to the factor of trust as being able to depend on someone or something for the long term generates trust. Whereas accountability focuses on taking responsibility for actions, dependability focuses on a consistent attitude of trust. This is a complex element as different components of a system have to pass reliable information:

*"This disaster could have been prevented if, instead of continuing to create more and more complex planes, we stuck to a simpler design and just streamlined it from there. Modern planes have become too complex and have too many bells and whistles. By designing the plane to need auto-corrective software, we made it so that the planes would essentially try and fly themselves based on various different sensors. Depending on what those sensors detect, the plane could either have a smooth and safe trip, or plummet into the ground based on false information from the sensors. By having the new Boeing Max planes be reliant on the software that auto-corrects the angle of the plane and pushes the nose down, you have essentially put another sensor into the mix that could potentially be fatal. If we had designed the plane in a way that would have not needed the use of this software, or at least been transparent about what software had been newly installed, then most likely this disaster would have never happened." [Student 6]*

*"In order to ensure future safety, transparency, and trust, the solution is for Boeing to slow down on its desire to rush its design of aircraft, not pressuring the FAA to speed up regulations, and for the FAA to have more funding. If Boeing slowed down its design process, they could create solutions to any problems it faces, instead of temporary solutions such as the MCAS system. This would in return ensure more safety." [Student 9]*

c) **Bias:** The problem of bias is becoming commonplace as software – algorithms – is used to make decisions that impact people directly. Based on datasets that themselves can be biased, software-based decision-making can affect people negatively, and these concerns are present in the facial recognition, credit and lending cases. Through the role-play discussions and the readings, students learned about the issue of bias and also debated ways to mitigate the problems.

*"From an ethical standpoint, I learned that bias occurs frequently and that when it is involved in real-life situations such as credit lending or court cases, we should do our best to mitigate it. [The lending case] showed me the importance of bias which I will always take into account when AI based credit lending becomes the norm." [Student 21]*

d) **Diversity:** The ability to include a diverse population and be inclusive in terms of the workforce and the user population, is critical for development of software that serves everyone. If addressed correctly, diverse representation can address bias to some extent. Consequently, diversity is a core

ethical concern for software engineering. This issue was central to the facial recognition and lending and credit cases and students identified and discussed them:

*"For the Lending/Credit/Loan case study, I learned that there should not be any biases within the algorithm that can influence the application's decision. Also, alternative data as a factor can be unethical, such as the application being denied based on the application's demographic and socio-economic. According to my previous research, while most banks have regulation and transparency on how they approach their decision for every application, they lack diversity which I believe is unethical. Everyone deserves the same chance when financial institutions are deciding on loan applications." [Student 39]*

*"Facial recognition (FR), in its current form, cannot be used on a diverse racial and economic student body. FR systems have inherent bias stemming from the training sets used by many companies. A recent study, conducted by the National Institute of Standards and Technology (NIST), found that facial recognition algorithms were between 10 and 100 times more likely to misidentify Asian and African Americans. These stark differences in FRs accuracy show how primitive and dangerous implementing this technology could be. Imagine a scenario where minorities at the university are regularly harassed by our security, despite completing the health check, because of the FRs inaccuracies. We must create a safe environment for students and faculty. However, we must also work to create an environment that does not further entrench racism and bias in our community. The current system proposed not only discriminates with its use of FR but also its reliance on smartphones. With COVID-19 leaving many students and their families without a stable income, it is doubly immoral for the university to assume access to these devices." [Student 18]*

e) Privacy: The impact on privacy of new forms of software is one of the most recognized ethical impacts and students brought it up during the facial recognition discussion.

*"There are benefits to using facial recognition on university grounds; however, it does not outweigh the consequences. Our top priority should be protecting the university's students' safety and privacy during COVID and after. Using facial recognition at AHU will not protect the students' safety or privacy, especially our students of color. Instead of incorporating a biased system that will disproportionately affect students of color, we should carefully follow and enforce CDC guidelines on campus as we have been the past year." [Student 4]*

f) Safety: Safety was often a critical element of the cases, in particular the Boeing case where the disaster resulted in very high number of casualties.

*"While learning about the Boeing incident, I understood how a seemingly minor issue (such as competing with another company for sales), could potentially put others at risk. Boeing is a prime example of why you shouldn't skim on quality or time when it comes to something as*

*serious as the lives of 350 people. The lessons on ethics learned with this scenario are to always be honest and transparent with others and to not prioritize financial gain over the safety of others." [Student 27]*

Safety was also present in another form where the use of technology would make the users safe, e.g., in the case of using facial recognition on campus. For students, the issue of safety was interlinked with other elements, and they saw it both as a non-debatable element, but also as a trade-off.

*"We came to the conclusion that everyone valued and needed their privacy. For this reason, we were against the idea of using facial recognition. However, we recognized that something must be done to ensure the safety of the college. So, we came up with an alternative solution, which was to examine everybody directly instead of using technology that would only violate our privacy." [Student 26]*

g) Sustainability: Environmental aspects of information technology are increasingly coming into focus with the climate crisis and sustainability is not considered an integral aspect of ethical and responsible software engineering. The Volkswagen case study brought this issue into focus for the students.

*"The Volkswagen case study showed me that even the most respected companies are capable of behaving unethically. Before this scandal, Volkswagen was one of the most well-respected car companies. Cheating car emissions tests is completely unethical, and extremely bad for the environment. This case study shows me that we need to hold companies to an ethical code, to ensure that they don't do any unethical business practices." [Student 28]*

The students realized that the effect on the environment of software use does not necessarily have to be direct but indirectly, software can play a central role by impacting how a system is checked or verified.

*"Because my role wasn't very ethical, I could view the situation from two completely different perspectives. I'd say the main lesson on ethics in this scenario is that even if you aren't necessarily putting people in harm's way (like Boeing, for instance), deceiving customers is still unethical. Additionally, Volkswagen was still harming the environment. I think being open and honest with others is important, especially when running a business." [Student 27]*

*"In the Volkswagen scenario, I learned that no matter what I try to do to make my products seem better, it is unethical to try and cheat the system when it affects the environment. No matter if a company is trying to make a few extra dollars or they are trying to make a tight deadline, installing a device that purposefully changes the emissions to meet the standards, and then after the inspection to raise the emissions to make the vehicle output more power, is not the ethical thing to do in any situation. If it was on a smaller scale then I believe that VW would not have gotten the flak that it did, but this emissions issue affected vehicles across the globe." [Student 6]*

*h) Surveillance:* The ability to surveil people has increased dramatically with new technologies, including digital cameras and software that can process the captured video. Especially in terms of surveillance through facial recognition, there is a need for increased awareness, and through the case study related to the use of facial recognition on a college campus, students both became aware of this topic and discussed it from different viewpoints.

*"The main criteria when choosing alternative options to facial recognition was really that we are opposed to the idea of surveillance and our data potentially being sold to third parties we are unaware of. This is why we agreed on COVID testing and temperature checking over facial recognition." [Student 27]*

In addition to the proposed software implementation, students also discussed their personal experience with the technology, including their lack of understanding of how some of this software works and the need for more transparency.

*"With privacy and facial recognition, students are just not going to be aware of the extent this technology will be used. We're already concerned about facial recognition tracking software for our exams, so if we have constant surveillance that impedes our privacy like this, it will just not be very good for the mental health and well-being of students. The other issue is the bias and discriminatory problems that could inhibit the facial recognition technology. We don't know if the AI will target certain demographics, like people of color, so it's a questionable tech." [Student 12]*

*i) Transparency:* Being communicative and clear about a process or why a change is being made leads to people outside of the business being able to perceive what is happening with a software or process. Transparency in software development does not mean total visibility of the internal components. Transparency can be encouraged between the business and users, developers and users, or among other stakeholders. Especially when navigating business concerns, the cases highlight transparency's ability to highlight other business values.

*"Boeing was not transparent with its pilots or the public. There was little to no communication within the company and it resulted in a lot of lives being lost. The public lost a lot of trust in this company due to the lack of transparency." [Student 25]*

*"The best approach is to make the AI, but make it public, or be able to show it to the people you are using it for. With discrimination and bias it can be bad, but Kwame is in the middle. He knows the ins and outs of AI and would make it so that it does not have an ounce of bias in it, not with demographics or skin color. He would work to make it transparent, which is what it needs." [Student 33]*

*j) Business concerns:* Business activities often raise ethical questions or concerns that, if not addressed, can result in issues escalating. Engaging with the cases and the discussion that follows raises many of these concerns and highlights the

sometimes opposing values of business activities that result from a decision.

*"From my perspective as Mary Bradley, I would say out of experience and for the time I spent in the company, the disaster happened due to the competition between them and Airbus, and how the build of the airplanes was rushed and didn't provide the safety needed. It could've been prevented if they gave themselves more time to build this project and be more careful and safer and not worry about the competition." [Student 10]*

*k) Work concerns:* Organizational culture often determines how employees act and what values they prioritize and it is another important aspect of ethics that students need to be aware of, and learned through their discussion:

*"The company culture at Boeing proved to be their downfall. A breakdown in collaboration and business executives caused over 300 people to die. Greed motivated the company to cut corners and prioritize good business over good engineering...The VW role-play, like the Boeing role-play, shows how a breakdown in communication can lead to compromised ethics within an organization. Like Boeing, VW business executives were unwilling to listen to their engineers and as a result, cut corners. These cut corners destroyed their reputation and the environment." [Student 18]*

## B. Situated Factors

*a) Complexity and Contradiction:* As highlighted in the discussion of the aspects of ethical and responsible software discussed in Table 1, many of these issues occur concurrently. This leads to situations where there is a lot of information to process for a decision to be made addressing the level of complexity in the case.

*"All members of my group realized that the Boeing Max disaster wasn't the fault of a single person, but rather a combination of events by all parties involved. The group also realized that moving forward, the only solution is for everyone involved to be more open to each other and be willing to work with each other." [Student 9]*

*"The ethical oversights made by Volkswagen include company culture and transparency. Their company culture was a toxic work environment where engineers did not feel comfortable bringing up their concerns to management. Management promoted a mindset where it must get done no matter what, and if you cannot do it, someone else will. They also lacked transparency to the consumers and the EPA on the emissions that this car produced, which caused more harm to the environment. Their response to the crisis was handled quite well. They issued refunds for all cars with the problem, the CEO resigned, and they continued with transparency through the entire process." [Student 1]*

There is also room for contradiction, as the roles encourage students to think about the cases in different ways, and students will often come up with many smaller micro-decisions leading toward a larger decision about the whole case. Navigating the values laden in these micro-decisions outlines the contradictions as they navigate the ideas.

*"Volkswagen failed to demonstrate a strong corporate social responsibility. They ignored emissions regulations earn higher profits. The emissions rates were damaging to the environment and people, who could develop cancer from the smog produced. I have a mixed opinion of Volkswagen's response to the crisis. One thing that I think was done poorly was the lack of accountability. Volkswagen committed to transparency, however, continued conducting shady practices such as installing hidden devices in their cars. One thing that I think was done well is Volkswagen replaced their CEO. The new replacement promised deep reforms and a focus on developing electric cars." [Student 24]*

*"It taught me that we do not always think about how technology will affect average people and how it will benefit the people implementing them. The facial recognition technology had racial bias and other factors that would negatively effect people on the campus, but it also seemed to have lots of benefits it would provide to the university." [Student 17]*

b) *Human Factor in Automation:* Students recognized that although the use of technology in many cases was inevitable given its potential usefulness, they did realize that to ensure that users were not harmed unnecessarily or unintendedly, a human-in-the-loop approach was needed. Algorithm-driven systems need to work in conjunction with humans.

*"I think we need to work with decision-making algorithms side by side. The modern decision-making algorithms are good but they need human supervision. Their decisions can't be trusted completely. The final decision of these algorithms needs to be double checked by humans working on the same project. Hence, in the present time I think the best approach to make a decision on the loan application is by working with both algorithms and humans. I don't see any barriers to this approach if we work both with the algorithms and humans combined." [Student 30]*

c) *Limits on Technology Use:* Finally, for some students, the ethical decision was not to use technology at all, especially if there was no way to control some of the factors and downsides associated with the use of a system:

*"My group came to the conclusion that using advanced technology is not always the solution. There must be a purpose and a real need to use it, otherwise it would cause more harm than good. It is important for people to understand what this technology brings to the table and what the pros/cons are of using it. While using FR might be useful and a good way to keep the students on top of the COVID-19 situation, it is only a short-term solution. What would this technology be used for after the fact, or where will all the stored data go? These questions must be asked by everyone because it is ultimately up to us to have a say in what we do as a community on campus." [Student 22]*

### C. Overall Learning Related to Ethics

Finally, we report overall learning related to ethics among students. For many students, this was the only course related to ethics they took and the only time they were exposed to case

studies. Therefore, both the topic and the pedagogical intervention were new to them.

*"Before taking this class, I never truly gave ethics a lot of thought. I have deep-rooted moral principles that I have always followed, but have never truly related those principles to the real world environment, specifically in a professional environment. After taking this class, you have opened the door to deep thought about ethics in the professional world. I have discovered new principles that will help me later in my career." [Student 23]*

Most students would go on to have a career in the information technology industry, starting at entry-level and working on and with different software.

*"As I am graduating in a couple of weeks, this course was one of my favorite courses I have taken at [the university]. As a vegan and intersectional activist, ethics have always been really important to me. In my years studying IT, it did not feel fulfilling because it felt like I wasn't doing anything impactful for society. However, after this course, and especially during the lending/loan topic, I feel inspired to work in a field and create things that prioritize doing good for society while still being able to pursue IT." [Student 4]*

*"I learned that AI can be biased, which made no sense to me at first. I found it crazy that things like AI could have bias, especially since it is literally a computer system. I also didn't realize that there was a certain type of bias regarding loans and lending. I know that credit history and all that matter when pulling a loan, but to see a computer deny people was surprising. Hopefully, in the future, AI won't dominate the world." [Student 33]*

For many, the first tasks would be programming-related. Consequently, the topic of software development was of inherent interest to them.

*"After learning the extent to which Volkswagen went to cheat emissions tests, I learned that all companies, especially tech, have the ability to cheat or alter the way their software or information works because they are the creators of this sophisticated technology that we have zero knowledge of on the inside. For example, VW hired skilled programmers to implement complex code or Apple secretly underclocked or slowed down iPhones to fix battery issues. It's something to consider, especially with new and upcoming advancements." [Student 21]*

Interestingly, preparing for their roles motivated students to undertake extra research and learn more about the topic beyond what was needed for the case discussion, often leading to strong insights about the need for learning about ethics in relation to software systems.

*"The facial recognition role play sent me down a rabbit hole of articles. Researching this, I found out how much other countries invade their citizens' privacy. These countries do not care for ethics when using facial recognition tools.... It shows how much ethics are important in anything regarding technology and implementing into human lives." [Student 31]*

## VI. DISCUSSION

The increased use of software across all aspects of lives and society has created a need for software engineering ethics. This need is especially acute as new forms of algorithms and AI-based systems become common. Based on the findings from this specific study and also drawing on lessons from using role-play case studies across ten different course offerings [43-44], we now discuss the relevance of the findings.

**Identifying Ethical Concerns:** In this research study of the use of role-play case studies for teaching software ethics, we found that students could identify different elements of ethical concerns and develop a more nuanced understanding of how software shapes ethics on the ground. Regarding different concerns related to ethics, the cases allowed students to recognize and reflect on the values of transparency, trust, safety, privacy, and so on. The concerns they identified are not exhaustive but are vital to begin thinking about ethical and responsible software engineering.

**Situated Aspects of Software Use:** In addition to identifying specific concerns, in relation to the pedagogical approach advance here, the students could look at the situation in a contextual manner and understand the complexity inherent in software systems once they are deployed. They further identified and discussed the limits to the use of technology and the need for human mediation when systems are deployed, especially about automation. Overall, students displayed a remarkable understanding of ethical issues related to software and evaluated the course highly in terms of their learning. The cases also gave them a view of the professional world of software engineering, going beyond the knowledge they had acquired in their courses.

**Active Learning and Self-Regulation:** Consistent with prior work on active learning, we found that when students are asked to engage with the material with the aim to use that knowledge for a discussion, they show a deeper understanding and increased motivation. This was facilitated also by the use of different forms of content – readings, videos, podcasts – and the freedom students were given to do their own research online. The diversity of content allowed students to find ways of learning that worked for them, and as they researched more, the more interesting content they came across. In other words, self-regulation and autonomy, as the literature says, are important factors in student learning, and encouraging that through pedagogical interventions is important. In addition to the active learning component, the focus on role-play provided students both guidance and guard rails in terms of how to engage in a discussion and what to say, but also make them reflect on the different stakeholders involved in the process. This diversified their viewpoints and sometimes led to a change in their perspective.

**Novel Aspects of Pedagogical Approach:** There was a novelty factor as well both in terms of content and the use of case studies and role-play discussions as students had not yet encountered this form of pedagogy. In relation to the content of the course, the topic and the readings allowed students to engage their whole self, their identity, in the topic and discussion as many of them were aligned in their values with

their roles and with the course topic in terms of creating more ethical and responsible software. In terms of pedagogy, in all their other courses, they were taught in a traditional manner with lectures and if there was group work, it was not directed in any form but was more of a “work as a team” approach. This often resulted in free-riding but in role-play discussion it is difficult to free-ride as everyone has to contribute. Student participation varied, but they all had to participate.

**Teaching Insights for Using Role-Play Case Studies:** There are several lessons from this study for using role-play case studies for teaching software engineering ethics: 1) The cases should be designed to be relevant for students, this means either newsworthy topics or those they can relate to at a personal level. In our case, Boeing and VW were both newsworthy and facial recognition and lending are relevant to students’ everyday lives. 2) The scenarios should be succinct and leave space for interpretation by students so that they have to struggle with a response and it is relatively open-ended. This makes for a better discussion. In our case, teams often reached recommendations that were in opposition to other teams. 3) Although our findings here are from a single course offering, we realized there is value in creating a simpler form of a case initially and extending it over time with more details if students show an interest in it and there is evidence of learning. 4) Multiple assessment methods are possible for case studies and should be implemented. We used pre/post discussion questions, concept maps, and, in some cases, multiple choice questions, and all are feasible for this approach.

**Faculty Effort and Resource Issues:** Like most active learning pedagogical strategies, designing and using role-play case studies is a time-consuming intervention. There are several ways though to mitigate the effort required. First, many cases that have been developed by others are available publicly with permission to re-use [43-44]. Second, given the impact this approach has on students and their learning, even a short case and even a single case can contribute to raising ethical awareness among students. Third, once the initial development has been done, the approach is quite flexible and can be easily adapted to other courses and modules if required.

## VII. CONCLUSION

In this paper, we present a study of the efficacy of using role-play case studies for teaching software engineering ethics. We found that by participating in this pedagogical intervention students were able to learn about different ethical concerns and about contextual factors affecting ethical use of software. We provide details of student assessment. Case studies and other materials developed as part of this project are available online publicly through open access for others to use.

## ACKNOWLEDGMENT

We thank the study participants and the U.S. National Science Foundation Award # 1937950, 2335636 for partial support. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the funding agencies.

## REFERENCES

- [1] C. O'Neil, "How algorithms rule our working lives", *The Guardian*, Sep 2016.
- [2] F. Pasquale, "The Black Box Society: The Secret Algorithms That Control Money and Information", Cambridge, Massachusetts; London, England: Harvard University Press, 2015.
- [3] C. Fiesler, N. Garrett, and N. Beard, "What Do We Teach When We Teach Tech Ethics? A Syllabi Analysis," in Proceedings ACM SIGCSE, Feb. 2020, pp. 289–295.
- [4] N. Garrett, N. Beard, and C. Fiesler, "More Than 'If Time Allows': The Role of Ethics in AI Education," in Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society, Feb. 2020, pp. 272–278.
- [5] A. Narayanan and S. Vallor. "Why software engineering courses should include ethics coverage". *Commun. ACM* 57, 3, 23–25, 2014.
- [6] G. Génova, M. R. González, and A. Fraga, "Ethical Education in Software Engineering: Responsibility in the Production of Complex Systems," *Sci Eng Ethics*, vol. 13, no. 4, pp. 505–522, Dec. 2007, doi: 10.1007/s11948-007-9017-6.
- [7] J. Hughes et al., "Global and Local Agendas of Computing Ethics Education," in Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education, Jun. 2020, pp. 239–245.
- [8] B. B. Bullock, F. L. Nascimento, and S. A. Doore, "Computing Ethics Narratives: Teaching Computing Ethics and the Impact of Predictive Algorithms," in Proceedings of ACM SIGCSE, Mar. 2021, pp. 1020–1026.
- [9] J. P. Hertel and B. J. Millis, *Using simulations to promote learning in higher education: an introduction*. Sterling, VA: Stylus Pub., 2002.
- [10] D. Gotterbarn, K. Miller, and S. Rogerson, "Software engineering code of ethics," *Commun. ACM*, vol. 40, no. 11, pp. 110–118, 1997.
- [11] D. D. Gotterbarn and D. K. W. Miller, "Computer ethics in the undergraduate curriculum: Case studies and the joing software engineer's code," *Journal of Computing Sciences in Colleges* 20, no. 2 (2004): 156–167.
- [12] D.D. Gotterbarn. "Software Engineering Ethics," 2001. Encyclopedia of Software Engineering, Wiley Online Library." Available: <https://onlinelibrary.wiley.com/doi/abs/10.1002/0471028959.sof314>
- [13] ACM, "Code of ethics," 2018, <https://www.acm.org/code-of-ethics>
- [14] IEEE, "Code of ethics," n.d. <https://www.ieee.org/about/corporate/governance/p7-8.html>
- [15] Software Engineering Code of Ethics and Professional Practice: <https://ethics.acm.org/code-of-ethics/software-engineering-code/>
- [16] B.W. Boehm, *Software Engineering Economics*, Prentice Hall, NJ. 1981.
- [17] Y. Lurie and S. Mark. "Professional ethics of software engineers: An ethical framework." *Science and engineering ethics* 22 (2016): 417-434.
- [18] F. B. Aydemir and F. Dalpiaz, "A roadmap for ethics-aware software engineering," in Proceedings of the International Workshop on Software Fairness, in FairWare '18. New York, NY, USA: Association for Computing Machinery, May 2018, pp. 15–21.
- [19] R. Alidoosti, P. Lago, and M. Razavian, "Ethics in Software Engineering: a Systematic Literature Review". 2022, Technical Report. <https://research.vu.nl/en/publications/ethics-in-software-engineering-a-systematic-literature-review>
- [20] A. Rashid, K. Moore, C. May-Chahal, and R. Chitchyan, "Managing Emergent Ethical Concerns for Software Engineering in Society," in 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, Florence, Italy: IEEE, May 2015, pp. 523–526.
- [21] J. Singer and N. G. Vinson, "Ethical issues in empirical studies of software engineering," *IEEE Trans. Software Eng.*, vol. 28, no. 12, pp. 1171–1180, Dec. 2002, doi: 10.1109/TSE.2002.1158289.
- [22] J. Greeno, A. Collins and L. Resnick. "Cognition and learning". In R. Calfee & D. Berliner (Eds.), *Handbook of educational psychology* (pp. 15-46). New York: MacMillan, 1996.
- [23] J. Lave and E. Wenger. "Situated Learning: Legitimate Peripheral Participation". Cambridge University Press, 1991.
- [24] A. Johri, B. Olds and K. O'Connor. "Situative frameworks for engineering learning research". In Johri, A. & Olds, B. (ed.). *Cambridge handbook of engineering education research*, 47-66, 2014.
- [25] A. Johri and B. Olds. "Situated engineering learning: Bridging engineering education research and the learning sciences". *Journal of Engineering Education* 100 (1), 151-185, 2011.
- [26] R. K. Sawyer and J. G. Greeno, "Situativity and Learning," *The Cambridge Handbook of Situated Cognition*, pp. 347–367, 2001.
- [27] C. van de Sande and J. G. Greeno, "Achieving Alignment of Perspectival Framings in Problem-Solving Discourse," *Journal of the Learning Sciences*, vol. 21, no. 1, pp. 1–44, 2012.
- [28] D. Martin, G. Bombaerts and A. Johri. "Ethics is a disempowered subject in the engineering curriculum". *Proceedings of SEFI* (2021).
- [29] J. R. Herkert. "Engineering ethics education in the USA: Content, pedagogy and curriculum". *European Journal of Engineering Education*, 25 (4), 303 – 313, (2000).
- [30] J. L. Hess and G. Fore. "A Systematic Literature Review of US Engineering Ethics Interventions". *Science and Engineering Ethics*, 24(2), 551–583 (2018).
- [31] J. R. Herkert. "Ways of thinking about and teaching ethical problem solving: Microethics and macroethics in engineering." *Science and Engineering Ethics*, 11(3), pp.373-385 (2005).
- [32] V. Varma and K. Garg, "Case studies: the potential teaching instruments for software engineering education," Fifth International Conference on Quality Software (QSIC'05), Melbourne, VIC, Australia, 2005, pp. 279–284, doi: 10.1109/QSIC.2005.18.
- [33] M. C. Loui. "Fieldwork and cooperative learning in professional ethics." *Teaching Philosophy*, 23, 139-156, (2000).
- [34] M.C. Loui. "What can students learn in an extended role-play simulation on technology and society?" *Bulletin of Science, Technology & Society*, 29(1): 37-47 (2009).
- [35] R. H. Prince, "Teaching engineering ethics using role-playing in a culturally diverse student group," *Science and Engineering Ethics*, vol. 12, no. 2, pp. 321–326, 2006.
- [36] C. Shaw. "Using role-play scenarios in the IR classroom: An examination of exercises on peacekeeping operations and foreign policy decision making." *International Studies Perspectives*, 5(1), 1–22 (2004).
- [37] D. A. Tamburri, M. Razavian and P. Lago, "Teaching software design with social engagement," 2013 26th International Conference on Software Engineering Education and Training (CSEE&T), San Francisco, CA, USA, 2013, pp. 61-69, doi: 10.1109/CSEEET.2013.6595237.
- [38] T. Kubica, R. Peine and I. Braun, "Role-Based Group Formations and Interactions to Foster Collaboration in Different Classrooms," 2020 IEEE 32nd Conference on Software Engineering Education and Training (CSEE&T), Munich, Germany, 2020, pp. 1-2.
- [39] R. G. Epstein, "The use of computer ethics scenarios in software engineering education: The case of the killer robot," in *Software Engineering Education*, J. L. Diaz-Herrera, Ed., in *Lecture Notes in Computer Science*. Berlin, Heidelberg: Springer, 1994, pp. 429–437.
- [40] J. Gogoll, N. Zuber, S. Kacianka, T. Greger, A. Pretschner, and J. Nida-Rümelin, "Ethics in the Software Development Process: from Codes of Conduct to Ethical Deliberation," *Philos. Technol.*, 34, (4):1085–1108, 2021
- [41] C. Marshall and G. B. Rossman. *Designing qualitative research*. 2014. Sage publications.
- [42] A. Strauss. *Qualitative analysis for social scientists*. Cambridge university press, 1987.
- [43] A. Hingle and A. Johri. "Role-Play Case Studies to Teach Computing Ethics: Theoretical Foundations and Practical Guidelines." *Proceedings of HCISS 2024*.
- [44] A. Hingle, H. Rangwala, A. Johri, and A. Monea. "Using role-plays to improve ethical understanding of algorithms among computing students." In 2021 IEEE Frontiers in Education Conference (FIE), pp. 1-7. IEEE, 2021.
- [45] A. Johri. "George Mason Tech Ethics," 2024 <https://onlineethics.org/collection-detail/George%20Mason%20Tech%20Ethics>

# In the Footsteps of a Professional Software Engineer: Exploring Role-play in Teaching Software Design

Fatma Meawad

*Chair of Software Engineering*

*Open Institute of Technology*

fatma.maeawad@faculty.opit.com

**Abstract**—Software design is an inherently abstract field that requires experience to be fully grasped. The considerable variations in how the software industry practices it further contribute to the lack of widely accepted pedagogy or instructions for it. This paper introduces a novel teaching methodology that places students at the heart of real-world software design scenarios. This approach involves simulations that mirror the career progression of software engineers, including the interview process and subsequent professional experiences. Students engage in role-playing exercises, assuming various roles within a company to tackle the practical challenges of scaling and refining a legacy software project. This experience is designed to be immersive by reflecting realistic conditions and responsibilities encountered in the software industry. This paper presents the detailed design of the scenarios and concludes with a detailed reflection on the effectiveness of this method from students' and instructor's perspectives. The results provide valuable insights into the impact of real-world simulations in sparking students' interest in the topic and offer essential guidance for instructing it.

**Index Terms**—experiential learning, real-world scenarios in education, software design education, role-play in education

## I. INTRODUCTION

Teaching software design is critical in shaping the capabilities of future software engineers [27]. Research in academia and industry reports that most software engineering graduates lack the skills needed to apply good design principles, engage in modeling, or communicate software systems effectively, particularly when analyzing various design trade-offs [2], [4], [10]. Eric Brechner, the Director of Developer Excellence at Microsoft, identified software design analysis as the most important topic among five essential topics required to close the gap between academic learning and industry demands [4]. Furthermore, with the rise of generative AI and coding assistants, the unique human capability to make design decisions about software construction becomes increasingly valuable. Generative AI, despite its advancements, struggles with the abstract nature of software design, highlighting the growing demand for graduates adept in these skills [6].

Despite its importance, teaching software design is a well-recognized challenge for educators. Design problems are "wicked problems" [3]. They often involve numerous possible approaches with different levels of detail. Moreover, software

design practices in the real world can vary greatly, contributing to the absence of a universally agreed-upon teaching approach [10]. Truly understanding and valuing the lessons of software design typically requires some practical experience and a commitment to creating long-lasting software products.

In my experience teaching software design to university students and training software engineers in the industry, I have seen a clear difference in how both audiences perceive the topic. Professional software engineers show real curiosity and value the topic often because they have dealt with complex, poorly designed software that's difficult to maintain. On the ground, their ability to plan and make related decisions in meetings directly impacts their career. In contrast, students typically do not prioritize the topic beyond achieving the grades. The question then arises: How to create an engaging learning experience for this subject in a university environment where students primarily focus on grades, and their project engagement is usually short-term?

Experiencing the distinct difference between the two audiences, university versus industry, led me to rethink how I teach this topic. I shifted the motivation of the module from emphasizing the project's future to the students' own futures and careers after graduation. While both elements exist, the student's career is at the center of the module narrative. In this paper, I report on my experience teaching software design through simulating real-world situations that early-career software engineers often encounter. I create a narrative where students are the main characters, following the path of a professional software engineer. The story starts with students interviewing for a software engineer III job at a company and continues till they scale one of its internal products. The scenarios are designed to help students experience how to incrementally refactor the design of a legacy project and communicate their change proposals to different stakeholders. As the instructor, I take on multiple roles in the story to bring the scenarios to life. The paper discusses the details of the experience and concludes with reflections on students' feedback and the lessons learned.

## II. RELATED WORK

The difficulties associated with teaching software design, as well as the necessity of its effective integration into a software engineering program, have led to extensive academic research dedicated to this area [10]–[12], [27], [29]. A detailed literature review reveals the diversity of the pedagogical approaches adopted while noting the prevalence of active and experiential learning methods [27]. Across the variety of existing approaches, integrating real-world practices stands out as the most appealing strategy to prepare students for the software industry. However, this strategy often comes with considerable costs in terms of resources, time, and complexity [13], [17].

Approaches that aim to mimic the real world range from highly authentic, such as directing students to contribute to active open-source projects [14], [29] and engage with real external stakeholders, to more basic approaches, like referencing industry standards and bringing in guest speakers from the industry [13], [27]. In all cases, the success of the learning experience fundamentally depends on using a sufficiently complex real-world project [23]. One notable thing missing from current software design modules is teaching students how to evaluate designs as they evolve. That is why it is advisable to involve students at this level in analyzing and refactoring the codebase of an existing project rather than initiating a completely new one from the ground up [2], [4], [27].

Several studies reported the success of guiding students' participation in live open-source software projects [14], [18], [24]. The open-source approach delivered on its promise by offering hands-on experience with real-world software engineering challenges and collaboration in large distributed teams. While providing valuable exposure to current technologies, this approach is highly demanding. It requires flexibility in navigating complex project settings, good programming skills, and engaging with various hard-to-reach stakeholders. Consequently, researchers are increasingly interested in investigating how to guide educators into integrating open-source contributions in teaching [25], [26].

Conversely, some instructors turned to role-play to avoid the challenges of the complex real-world setting. In this study, the authors simulated the dynamics of client-developer relationships by assigning interchangeable roles to student groups [17]. Instructors and students adopted various roles while working on the same project. This approach emphasized the development of competencies in areas like communication, requirement elicitation, and project management. Employing this method offers a practical insight into the challenges of project operations and engages students in a realistic scenario [22]. However, balancing roles and responsibilities in such dynamic conditions demands meticulous planning and a strong commitment from the instructors.

Reflecting on the diversity of related scholarly work, it becomes apparent that a hybrid approach, integrating active learning and hands-on, authentic experience of real-world industry practices, is essential. The degree of authenticity can

vary from full immersion to storytelling. With this in mind, the work presented in this paper is highly influenced by role-play and experiential learning practices [29]. In particular, the following practices are emphasized:

- Incremental refactoring of existing real-world projects with reasonable complexity.
- Using role-play to simulate real-world scenarios from a professional environment.
- Integrating continuous active feedback from both instructors and peers in the module's workflow
- Emphasizing non-technical skills and communication of various types of artifacts.

## III. MODULE DESIGN

### A. Context

The software design module at CODE University<sup>1</sup> is offered as an undergraduate-level elective with only two prerequisites: clean code and automated testing. The university strictly follows a project-based learning approach. Students from different programs across the university are free to work on their own unique projects with interdisciplinary team members. The projects are not predetermined or preapproved before students start a specific module. Alternatively, students choose the modules according to how their project learnings match the target learning objectives.

Student projects often extend beyond a single semester, with some evolving into startup companies. Consequently, project work assessed by instructors varies widely in technologies used, scope, domain, and complexity. In most modules at the university, participation in learning activities and related submissions are voluntary and primarily aimed at formative feedback. In addition, students have four chances to repeat the final assessment of each module while choosing the best attempt. This turns assessments into deep feedback sessions - adopting the assessment "for" learning versus "of" learning philosophy [28].

### B. Module's Objectives and Constraints

The software design module aims to introduce students to the key design principles and patterns at different levels of software architecture, focusing on their practical application in real-world projects. Another important objective is developing students' ability to identify, critique, and resolve design problems. Furthermore, students learn how to use various modeling techniques and diagrams to communicate software design at various levels of detail. A strong emphasis is placed on familiarizing students with industry standards for documenting and assessing software architecture, ensuring they gain relevant real-world skills.

Class attendance fluctuates significantly over time, ranging between 15 and 35 students. Considering that participation is voluntary and almost all students have part-time jobs, this is somehow expected but requires consideration in planning the activities. Asynchronous communication is always available

<sup>1</sup>code.berlin

through the module's Slack channel, where updates and todos are updated in a timely manner.

### C. Assessment

The only formal assessment in the module is a hand-in and a one-on-one oral examination at the end of the module. For the hand-in, the students get detailed instructions on the technical and non-technical artifacts that must be submitted. The artifacts are meant to help students demonstrate their achievement of the learning objectives in their own projects. The instructor has two weeks to thoroughly assess the submitted artifacts and prepare for a one-on-one with students to examine their knowledge further. This examination runs as an interview and focuses on assessing the student's ability to effectively communicate their design choices submitted in the artifacts.

### D. The Module's Background Story

Taking into consideration the module's objectives and with the intent of sparking students' curiosity about the subject, I structured the learning activities as the story of a professional software engineer experiencing the challenges and opportunities of this career path. The story involves a series of scenarios mimicking situations typically encountered by industry professionals, for example, undergoing interviews, making decisions in meetings, securing approval for proposals, and participating in workshops and peer reviews. Each scenario unfolds within the context of a fictional company seeking to recruit both junior and senior engineers. The company plans to extend an established legacy product that has been used for several years. Throughout this story, students become the hired team to analyze and refactor this legacy product.

Although the company in the story is fictional, I deliberately select a project from an active open-source project, ensuring it's not specifically tailored for this module. This approach is crucial for adding realism to the narrative and showcasing the real-world applicability of the principles taught within the module. In the two editions of this module, the company developed educational games, and the chosen project was a maze generation and solving application originally created for boot.dev<sup>2</sup>, an online educational platform. Even though the project selection is made without prior planning, certain criteria guide the choice; for example, ensuring the project offers a manageable codebase size, independence from specific frameworks, interesting use cases, and poor documentation.

### E. Pedagogy

The overall pedagogy for this module closely aligns with role-play or experiential learning within a project-based learning setting. The students and the instructor take on different roles and practice different related responsibilities in the whole setup. I relied on my background in the industry to recreate the conditions, artifacts, and even the humor that sometimes arises in certain situations. Furthermore, I interviewed two senior software engineers working in large corporate environments

to validate suggestions related to the technical documentation and the important topics to cover.

In each session, I use inquiry-based learning to further stimulate students' interest in the subjects. Discussions were held without explicitly naming the module's topics until the issues at hand were explored and resolved. This approach transformed the sessions into problem-solving dialogues rather than lectures on specific topics. For instance, students often tackle various challenges using design patterns without realizing they employ them. The revelation at the conclusion, where they connect the dots, significantly enhances the value of these practices.

The module's goal is not to exhaustively cover every topic but to explore enough examples, prompting students to explore or apply other concepts independently. For example, within the realm of Design Patterns, the scenarios were crafted to touch upon two or three patterns within three distinct categories, directing students to a comprehensive catalog of patterns for broader application. For assessments, students were not required to prove mastery over specific patterns but rather apply relevant ones to their project domains.

Finally, the refactoring and analysis approach was chosen because it demonstrated advantages in teaching this module rather than starting from scratch (the greenfield method). Over time, students get to observe the evolution of the same product as it adapts to the changing requirements and needs of different stakeholders. In the meantime, since students are assessed on their own university projects, several review workshops were added to the story as part of the company practices to accommodate this unique setting while keeping the story's theme. Combining the legacy product review with continuously evaluating students' own projects brought role-play to a mix of problem-based and project-based learning.

## IV. MODULE OUTLINE

Table I presents the outline of the software design module over twelve weeks, highlighting the details of the scheduled scenarios and the target module topics. There is a total of 8 synchronous sessions: six in-person, one hybrid, and one remote. This variety intentionally reflects the contemporary hybrid work environment commonly experienced by software engineers. The following subsections discuss the details of each scenario, the discussion, and the conclusion. At the conclusion of each session, the discussion learnings are mapped to specific topics in the software design module.

### A. Weeks 0-1: The Interview

**1) The Scenario:** A week before the first session, students receive an invitation from a recruiter for a second-round interview for a software engineer III position at an educational game development company. The interview will be a meeting with both business and technical stakeholders of a company's existing product. The primary objective of the meeting is to evaluate the product's maintainability and potential for further development, particularly the Graphical User Interface (GUI). The invitation provides tips on preparing to evaluate

<sup>2</sup><https://www.boot.dev/>

TABLE I

THIS TABLE PRESENTS THE STORY OF THE SOFTWARE DESIGN MODULE OVER 12 WEEKS, LISTING THE DETAILS OF THE SCENARIOS, THE CORRESPONDING DISCUSSIONS, AND THE TARGET MODULE TOPICS.

Week	The Scenario	The Discussion	The Conclusion: Module Learnings
0-1	<b>The Interview:</b> <ul style="list-style-type: none"> <li>Students receive an invitation from a recruiter for a second-round interview for a software engineer position.</li> <li>The invitation provides tips on preparing to evaluate the maintainability of the company's product and includes information on potentially leading a team.</li> <li>The interview consists of a meeting with the product's business and technical stakeholders.</li> </ul>	<ul style="list-style-type: none"> <li>The instructor alternates between business and technical interviewers, posing their respective questions to the students.</li> <li>After reviewing and analyzing the students' proposals, the instructor offers potential solutions specific to the company's product.</li> </ul>	<ul style="list-style-type: none"> <li>The module's learning Objectives</li> <li>The Motivation for Software Design and Quality Attributes</li> <li>The Big Ball of Mud Pattern and Maintainability</li> <li>Identifying and managing technical debt and the use of code analysis tools</li> <li>Agile modeling techniques</li> </ul>
2-3	<b>Design Meeting: Planned Enhancements</b> <ul style="list-style-type: none"> <li>The engineers receive a message congratulating them on securing the job.</li> <li>This message extends an invitation to join one of the company's squads and their Slack channel.</li> <li>The message also offers tips on preparing for the initial design meeting, where they will discuss the Features Roadmap with senior technical engineers.</li> </ul>	<ul style="list-style-type: none"> <li>The instructor takes on the role of the senior tech lead and welcomes the new candidates.</li> <li>The lead facilitates open discussions regarding students' proposed changes for planned enhancements.</li> <li>Following that, the lead showcases various design refactorings to enable the extensibility of the company's products and then proceeds to outline the planned technical modifications.</li> </ul>	<ul style="list-style-type: none"> <li>Loose coupling and cohesion</li> <li>SOLID principles and Extensibility</li> <li>Modelling and communicating code structure, static diagrams</li> <li>Documentation (TDDs) and Architecture Design Records (ADRs)</li> </ul>
4-7	<b>Design Meeting: Urgent Client Requests</b> <ul style="list-style-type: none"> <li>This scenario is set at a later time when the app has seen increased usage. The scenario covers two sessions.</li> <li>A message is sent to the engineers, informing them of an urgent design meeting due to clients' urgent requests, highlighting time and budget constraints.</li> <li>The message lists a set of requests needing prompt technical evaluation.</li> <li>The technical team must specify a solution and its consequences regarding time and technical debt.</li> <li>This scenario runs over two sessions (4 weeks).</li> </ul>	<ul style="list-style-type: none"> <li>The instructor switches roles between the senior tech lead and the product owner.</li> <li>The lead encourages open discussions on the squads' proposed solutions, with the product owner interjecting when proposals seem complex.</li> <li>Following that, the lead presents zero-modification solutions that satisfy the product owner, focusing on the extensibility of the company's products before moving on to planning technical changes</li> </ul>	<ul style="list-style-type: none"> <li>An Introduction to Design Patterns and their types</li> <li>In particular, applying the decorator, adapter, observer, wrapper, facade, state, strategy, and factory patterns.</li> <li>Assessing design alternative and their trade-offs</li> <li>Structural diagrams and behavioral diagrams</li> <li>Prioritizing and planning design decisions.</li> </ul>
8-9	<b>Design Meeting: The future of the application</b> <ul style="list-style-type: none"> <li>Assuming multiple years have passed, a message is sent to the students to discuss the future of the application.</li> <li>The message outlines the product team's roadmap and vision.</li> <li>Product management aims to scale up to a multi-user application, focusing on a collaborative learning experience and enabling interoperability with learning management platforms.</li> </ul>	<ul style="list-style-type: none"> <li>The instructor switches roles between the senior tech lead and the product owner.</li> <li>The product owner discusses the vision and gives examples.</li> <li>The lead presents various high-level architecture styles and prompts the engineers to discuss the best organization that fits the future vision of the application.</li> <li>After open discussions, the lead demonstrates clear refactoring steps of the application into one of those organizations.</li> </ul>	<ul style="list-style-type: none"> <li>Architectural Patterns and their quality attributes</li> <li>Clean &amp; ports and adapters architecture</li> <li>Domain Driven Design</li> <li>Event-driven architectures</li> </ul>
8,10, 11	<b>Design Review and Consultations</b> <ul style="list-style-type: none"> <li>Students receive invitations to internal events to exchange feedback on technical design documents.</li> <li>There are three invitations: two for informal reviews featuring open discussions and one for a formal review where the senior tech lead must decide whether the design is approved.</li> <li>The invitation includes detailed preparation tips on the artifacts needed for review.</li> </ul>	<ul style="list-style-type: none"> <li>Review meetings are formal and must conclude with a decision of whether the design is approved or needs revision.</li> <li>Instructor acts as the senior tech lead in both meetups</li> <li>Squads engage in presenting and reviewing others.</li> </ul>	<ul style="list-style-type: none"> <li>Technical design documents (TDD) and design review checklists</li> <li>Documenting, communicating, and evaluating models using the C4 technique.</li> <li>Preparation for hand-in and assessments</li> </ul>

the maintainability of the company's product. It includes information on potentially recruiting and leading a team of junior developers to advance this product.

2) *The Discussion:* In the actual session, the first 30 minutes are dedicated to introducing the subject of software design to the students, highlighting its significance to their careers as software engineers. I present the "The Big Ball of Mud" pattern and the complexities associated with working with legacy code. This introduction aims to shift the students' mindsets, preparing them for role-play and getting immersed into an industry-like experience. I also stress the distinction between the skills relevant to this module and basic programming skills, explaining why I have chosen Level III for the scenarios. Furthermore, I highlight the importance of non-technical skills, especially for this level, justifying the hint in the interview preparation about leading a team.

After the introduction, I step into the role of the technical interviewer and ask questions related to assessing the maintainability of the shared product's code, for example:

- Tell me about your approach to assessing the codebase maintainability
- What are your findings?

Subsequently, I transition to the role of a business team interviewer, inquiring about the product's future prospects with a question like:

- What are your recommendations moving forward knowing our plans for the product?

After each question, students can share any prepared proposal for answers or participate as interviewers and ask questions to other students. During this time, I avoid introducing any technical terms or principles.

3) *The Conclusion:* Once the open discussion is done, I revert to my role as the instructor and steer the discussion toward the module's key topics. I formally introduce the term maintainability, discussing methods to assess it and its industry-standard artifacts. The topic of technical debt is also explored, including strategies for its management (sizing and prioritizing the issues), alongside code analysis tools and their application. To solidify the learning process, I illustrate each topic with examples of the company's product, which is already closely examined by students at this point. A detailed analysis of the codebase's health is presented using results from various code analysis tools. I use commonly known tools like SonarQube [16], Codescene [7], and Pylint [20] while acknowledging the existence of many other tools in the field.

Lastly, I address the need to set up an efficient development environment, preparing for the integration of a new team of junior developers to facilitate smooth onboarding and collaboration. This highlights the significance of having proper technical documentation and the need to devise a strategic plan for managing technical debt. This aspect is demonstrated through the use of GitHub Wiki and Github Projects.

#### B. Week 2-3: Refactoring the Design

1) *The Scenario:* Following the interview and preceding the third week's session, students receive a message from the

recruiter congratulating them on landing the job. The message introduces students to their company's designated squad and their team's Slack channel. In addition, the message includes an invitation to a forthcoming meeting with the company's senior technical team to discuss the product roadmap. The following high-priority business requests are listed:

- Users can fully configure and run the maze from the command line.
- Users can configure the maze to work with different GUI toolkits, Tkinter, QT6, or others.
- Users can choose between different algorithms for generating and solving the maze.

The students are asked to prepare their proposals using Architecture Decision Records (ADRs) following the company's style guide attached to the message. Students have two weeks to prepare for this scenario. They have already closely investigated the maintainability of the code base and produced a list of prioritized refactoring tasks in the previous weeks. Furthermore, the students can discuss ideas over the team's Slack and prepare proposals until the meeting time.

2) *The Discussion:* At the start of the session, I take on the role of the lead senior engineer, welcoming the new hires and outlining the agenda for the meeting. I kick off the meeting by briefly introducing the planned enhancements. Then, I give the floor to each squad, allowing them to pitch their ideas for the requested enhancements. Every squad gets a chance to present their proposals, while all current members can provide feedback and engage in discussion. Squads can then vote on which proposal they want to pursue going forward. We assess the proposals based on the potential technical debt they may introduce and their flexibility for future modifications. The conversation at this stage is informal and open, aiming to capture all student suggestions within one hour.

After the open discussion, I take the lead again and present various visualizations of the codebase structure with (a) high-level package and class diagrams and (b) one detailed class diagram. I demonstrate using diagrams-as-code tools, such as PlantUML [19] and Mermaid [15], and diagram generation tools, such as GitUML [8] and Pyreverse [21]. After discussing the refactoring steps, I seek feedback on the most effective way to explain the code's structure. I aim to underscore the importance of conveying the code arrangement and its interconnections without strictly sticking to any specific notation. So, I only use terms like communication, the elements of code, and their dependencies. Then, with a broader perspective in mind, I prompt discussions about

- How do components depend on each other, and is it a good dependency?
- How does the size of the different components compare to one another?

Highlighting arrow directions helps unveil the dependencies between components while focusing on size brings out each component's distinct responsibilities.

3) *The Conclusion:* I wrap up this session by connecting the discussion points to key design principles in software

design, particularly focusing on coupling, cohesion, and the SOLID principles. I focus on the Single Responsibility Principle (SRP) and the Dependency Inversion Principle (DIP), which were explored through the directions of arrows and the sizes of components in our code structure discussion. Consequently, this has led to the following two steps for refactoring:

- SRP: Some classes notably took on more responsibilities than others, a problem that could also be spotted using code analysis tools. Breaking down the large components into smaller ones fixes the problem.
- DIP: We closely examined the relationships among components and questioned if important components depended on less important ones. It became clear that the application's core logic depended on a utility, which led to a violation of the DIP principle. Introducing abstracts to invert the dependency fixes this problem.

After applying the SRP and DIP principles, students see the diagrams of the updated code structure and the corresponding code. By the end of the session, students are often surprised to see how the codebase becomes more flexible just by adhering to simple principles and applying such easy steps. Applying DIP makes it straightforward to demonstrate the Open/Closed Principle (OCP), showing how easy it is to add different types of implementations using the existing abstractions. Moreover, using DIP facilitates the injection of dependencies into various parts of the code, enabling us to implement one of the requested enhancements smoothly without needing further changes. At this point, I introduce extensibility as a quality attribute.

#### C. Week 4-7 - Urgent Client Requests

1) *The Scenario:* By week 4, we assume that the new extensible version of the application has been released and more people are using it, leading to more feature requests. In this scenario, students get an invitation to an urgent client meeting to discuss the possibility of realizing a set of requested features under tight time and budget constraints presented by clients. In other words, engineers need to assess the features and pitch proposals for moving forward, indicating whether (a) no changes, (b) minor changes, or (c) major changes are needed. The meeting invitation lists the requests, which can be grouped as enabling users to:

- configure various logging channels to work concurrently.
- plugin external libraries for generating and solving mazes.

2) *The Discussion:* I begin the meeting by stressing the importance of keeping budget and time constraints in mind while evaluating the viability of any change proposals. Like in previous sessions, we continue with an open discussion to allow students to share their ideas informally and discuss alternatives. While going through each request and student proposal, I provoke students' curiosity about the impact of their changes, either in extra work to be done or technical debt for the future. Due to the nature of the scenario, I also give feedback as the product owner whenever any proposal appears

complex. I aim to timebox the discussion to 45 minutes or an hour maximum.

3) *The Conclusion:* The feature requests in this scenario are intentionally proposed to introduce design patterns to the students. When concluding the discussion, I present possible solutions requiring no changes to the existing code, only a few additions. At this point, students who have attempted to solve the problems are highly intrigued by the simplicity of solutions. At this point, I find the best opportunity to formally introduce the topic of design patterns, their categories, and the common use cases.

I present each solution using a static diagram showing the structure, a behavioral diagram showing the flow, and the corresponding code. For the particular requests in this scenario, I demonstrate how the Decorator and the Adapter patterns can meet all the new requirements. The power of applying the SOLID principles, particularly OCP, SRP, and DIP, is also emphasized since they enabled all the refactoring steps. In this session, the idea of evaluating alternatives is highly emphasized. Several patterns, such as the observer, strategy, and state, are also discussed as possible alternative solutions but with different consequences.

The discussion of design patterns took two sessions, covering four weeks of the module. The aim is not to cover all the patterns but to trigger the discussion and give the students the skills to choose and evaluate when to use them. Finally, I conclude the discussion that different domains have their own set of unique patterns and best practices. A key learning objective is to explore these patterns and apply them effectively.

#### D. Week 8-9: Roadmap and Planning

1) *The Scenario:* In this scenario, we imagine that multiple years have passed, and while there are not any pressing requests from clients, the product team is keen on planning the future of the application to keep up with current trends. The goal is to develop a multi-user application that supports a collaborative learning experience. Squads are asked to re-organize the code structure and present a proposal for a future structure to allow:

- Real-time multi-user collaborative learning
- Integration with online Learning Platforms

2) *The Discussion:* In this meeting, I switch roles between the senior tech lead and the product owner. The product owner discusses the vision and gives examples. Similar to other design meetings, squads can share their proposals for the application's structure and discuss how they can contribute to achieving the business vision of the product. Then, different squads vote on which proposal best suits the requirements. Without mentioning any architectural patterns, I ask if the structures the students shared are similar to any other existing applications to steer the discussion toward the topic.

The need to accommodate real-time communication and interoperation with different platforms helped introduce the concepts of (a) synchronous and asynchronous communication and (b) event-driven architectures. I present step by step

how relying on events can help decouple the application components from the start, enabling the easy integration of some form of message queue in the future. I summarize the discussion by comparing architectural styles and their varying quality attributes.

I take over most of the discussion in this scenario, even though the students start by giving proposals. Throughout the discussion, I observed students highly intrigued by how easy it becomes to move code components around and plug in new ones.

#### E. Week 10-11: Design Review Workshop

1) *The Scenario:* In the workflow of any engineering team, there is a process for reviewing technical design documents and change proposals. This review can happen asynchronously or synchronously. It involves different teams or several technical leads, especially for critical decisions. In weeks 8, 9, and 10, students are invited to join design review events as reviewers or reviewees. The company's technical design document template is also shared as a guide for the students with reference to example artifacts created throughout the sessions. In addition, The initial two review sessions are more informal, serving as consultations, but the invitation to the final session outlines strict rules for a formal design review workshop. This workshop is designed to mirror the seriousness of a meeting where outcomes are shared with external stakeholders and involve budget considerations. Thus, the submissions must meet rigorous standards and pass a strict checklist.

2) *The Discussion:* Before the meetings, the projects being reviewed are posted on Github or Google Docs. Students can start the conversation online by leaving comments or participating in Github Discussions. In the two informal meetups, I take on the senior technical lead role and consult students on generating useful artifacts to communicate their design decisions effectively. My approach involves asking more questions than providing answers, particularly focusing on the reasoning behind their decisions as reflected in their artifacts. I refer to the shared checklist, template, and the principles discussed throughout the series as much as possible. Students are also encouraged to serve as technical reviewers.

Furthermore, I prepare questions about challenges or best practices related to the unique setting of each student project under review, for example, asking about specific design patterns for smart contracts in Blockchain applications or Rust-based game simulations. In the documentation, each student must mention the top three quality attributes relevant to their use cases.

3) *The Conclusion:* Integrating design review in an engineering team's development workflow is a known leadership challenge. The practice highly varies in how it is applied, its level of detail, and its rigor. A detailed survey of existing architecture evaluation methods recommends a scenario-based evaluation technique with reference to a pre-defined checklist that matters to the team [1]. That is why an initial checklist is shared with the students at the first invitation for review.

Students are encouraged to contribute to the checklist by suggesting additions or modifications. In addition, I encourage students to explore tools, such as Codescene [7], to discover design violations. Relying on tools helps integrate the review step into a smooth workflow.

Since the story of the module stresses refactoring an existing project, we miss the discussion about following the process of documenting a project from scratch. That is why, in the review sessions, while evaluating students' own projects, I take the opportunity to emphasize the importance of having just enough technical design documents before starting to code. I promote the C4 (Context, Container, Component, Code) modeling technique [5] as an effective method to visualize various levels of an app structure and how they communicate it. The students have all welcomed this technique compared to any other visualization technique. It was particularly useful when visualizing a large product.

## V. REFLECTION

Incorporating real-world scenarios into teaching is expected to introduce uncertainty to a module's structure and plan. Therefore, I was prepared to leave room for change either in the scenario or in directing the discussions more effectively toward the module's learning objectives. Additionally, I actively sought feedback from students after each session through retrospectives and used short surveys during and at the end of the term. This section summarizes students' feedback and the lessons learned, highlighting future recommendations for adopting this approach.

### A. Students' Feedback

Overall, the feedback from students can be organized into several key areas: (a) requirements' clarity, (b) topics's scope and session structure, and (c) the value of the real-world scenarios and their relevance to their work experiences.

1) *Requirements' Clarity:* After each session, several students regularly expressed confusion regarding certain terms used in the messages they received from the company or about their expected tasks. For instance, they did not understand exactly what a "Software Engineer III" does or what counts as a 'minor or major change'. They thought these ideas were unclear and could mean different things to different people. I consider students recognizing this issue a successful outcome of the experience since this kind of discussion also happens in the real world. It's important for an engineer to know how to ask questions to get more information and set clear standards for their work. This is why, even after this feedback, I still used common terms in the industry and left it to the students to ask for clarification when facing any confusion.

2) *The story and industry relevance:* When asked about the story and scenarios in the class, students often described them as "great fun" and "interesting". They were curious about what would happen in each session. On multiple occasions, I had to interrupt the story to demonstrate an example not related to the main project. Students reported that breaking away from the story or showing extra unrelated examples was not helpful.

They discovered that these additional examples were not as effective in clarifying the principles as when illustrated using the company's product.

Some students mentioned they had experienced similar situations in their jobs and were glad to find relevant discussions at the university. The most relevant topic was the maintainability of legacy code. They all eagerly shared their challenges in class and were interested in the tools that could help them. The students were especially impressed by how using principles like DIP and SRP could separate different software parts, making it easier to manage. Some students reported that they began applying these tools in their workplaces to review their code and manage technical debt. Two students who have worked for more than two years stated that it is essential for all software engineering students to learn these topics before graduation, not as an elective subject. This reinforces the initial premise of this paper, namely that developers with industry experience develop a profound interest in software design.

3) *Covered topics and session plan:* Several students mentioned that the module focused too much on object-oriented design and should also cover functional programming. This is a valid and worthy point to consider for the future, as combining both methods is common in the industry nowadays. There was also feedback about the programming language used. Some students, especially Golang enthusiasts, did not think Python was the best choice. The disagreements and debates concerning the tools or technological stack chosen were highly beneficial in delivering a key learning objective in the module. This is because most of the principles and concepts taught are abstract and applicable across various technologies. Regarding the difficulty level, students expressed varied opinions on the complexity of the problems and the allotted time for solving them. Some suggested covering fewer topics for deeper understanding, while others advocated including more advanced topics and increased face-to-face discussions.

### B. Lessons Learned

Integrating role-play and real-world scenarios in the software design module has significantly improved my teaching experience, particularly as an instructor passionate about this subject. Observing the students' genuine engagement and participation in meaningful and challenging discussions mirrored my experience in industry. This gratifying experience has further reinforced my commitment to this teaching style with its blend of pedagogies in future iterations of the module. This section discusses the most successful practices and any relevant challenges and improvements.

1) *Refactoring a real-world project:* One important practice that worked well was using an existing open-source project as a legacy project for the hiring company. A real project, not specifically tailored for this module, has given a solid background to the story and the discussion. Furthermore, placing students as engineers handling a former engineer's old established code base enabled several interesting discussions around maintainability and extensibility where a company

aims to scale a product. This part of the story resonated well with students with any work experience since it is a common scenario.

The main challenges encountered are the time and effort required to refactor an unfamiliar code base in response to meaningful client feature requests. In addition, I needed to create alternative refactorings for each problem to discuss trade-offs and illustrate the effectiveness of the designated principles and design patterns. A sustainable approach to address this challenge would be incorporating these refactoring challenges into student capstone projects. It is a rich and interesting challenge for graduating students. Consequently, the output from these capstone projects could serve as the foundation for future iterations of the module. With this idea, it could be feasible to demonstrate the solutions across various programming languages and design paradigms.

2) *Adopting lean practices and automating documentation:* Software engineers often overlook the value of documentation, particularly diagrams of any kind, a trend also prevalent in the industry. The story places students in a position where they need guidance to understand an unfamiliar code base with few guiding resources. This situation led the students to recognize the need and request proper documentation to understand the code structure. In the interview phase of the story, all students highlighted this challenge as an obstacle to the product's maintainability, marking a noticeable shift in their attitude toward the importance of proper technical documentation in the rest of the scenarios.

To maintain the early interest provoked for documentation, I focused on teaching agile modeling principles and just enough documentation. With every refactoring step, I demonstrated how documents can be maintained as code using version control systems, diagrams-as-code, and various tools for automatically generating parts of the documentation. A specific notation was less emphasized when working with diagrams than the diagram's purpose and type (static or dynamic). The C4 modeling technique was the most successfully used among students, resulting in clear and useful artifacts. Moving the emphasis from merely acknowledging the need for documentation to seamlessly incorporating it into a productive development pipeline aligns more closely with the practices of software engineers and the real-world dynamics of their work environment.

3) *Bringing theory to life:* Software engineers value empirical evidence and tangible examples from code when discussing theoretical concepts. For instance, demonstrating how a new graphical user interface (GUI) can be seamlessly integrated into an application by applying the Dependency Inversion Principle (DIP) provides compelling evidence for the principle. That is why I provided a git branch for each refactoring step, illustrating the principles or patterns of transformation in code, diagrams, and ADRs. Aligning theoretical concepts and diagrams with actual working code is essential. This approach solidified my position in the sessions, as the discussions were no longer purely theoretical. The diagrams, principles, and patterns discussed took on a practical, real-world relevance,

encouraging students to appreciate and value these principles in a practical context.

Over time, I have realized that many students give up on good design simply because it is difficult to enforce in a team environment. That is why I was keen to introduce industry practices that would help them preserve a robust structure in their code base, particularly over time and within a team setting. In addition to integrating design review into the process, various tools are available to assist in identifying breaches of design principles. While these tools may not be as advanced as linting and styling tools, they are nonetheless numerous and useful. For example, I introduced a tool, import-linter<sup>3</sup>, to ensure proper dependency directions among packages, thereby preventing violations of DIP in their code structure. The tools can also be integrated into their development workflows. The students expressed considerable interest in such tools, as it was their first exposure to the possibility of automating such practices.

4) *Encouraging Co-creation:* Whenever possible, students were encouraged to participate in creating commonly shared artifacts, such as the design review checklist or the technical design document template. Before such artifacts are used as planned in the story, students are invited to modify or add to them. This approach helps provide a sense of ownership, enriching the learning experience. At the same time, such practice aligns with the collaborative practices found in contemporary agile software organizations. A better approach for the future is to share an incomplete draft of such artifacts to enhance the learning from this co-creation activity.

5) *Supporting an active online Community:* In addition to the need for adaptability in lesson planning, the instructor must actively participate in online discussions, mirroring typical communication in a company. Effective role-playing by the instructor is essential for the authenticity of the experience and for fully engaging students. To maintain such experience while decreasing the instructor's overhead, previous module participants can be encouraged to take various roles in the story. Another possible strategy is to involve the original developers of the open-source project in the story. They could contribute by suggesting feature enhancements and participating in reviews. While this may introduce additional complexity, it would likely enrich the overall quality of the learning experience while exposing students to the community.

Several students indicated that the time allocated for discussions was insufficient. Even though the aim is not to cover all the topics, I scheduled additional optional time slots to provide more room for extended discussions or feedback on design aspects. Additionally, I encouraged asynchronous discussions on topics that could not fit in the face-to-face session as much as possible. These practices not only address the time concern but are also standard industry practices, adding further authenticity to the experience.

<sup>3</sup>import-linter: <https://pypi.org/project/import-linter>

## VI. CONCLUSION

This paper described an innovative approach to teaching software design, merging role-play and inquiry-based learning within a unique project-based academic setting. Over 12 weeks, students were immersed in real-world scenarios typically encountered by professional software engineers, engaging them in incrementally refactoring a legacy software product. The objective was to spark students' genuine interest in the subject, a well-known educational challenge. With the hope of inspiring educators of the subject, the paper thoroughly discussed the complete narrative of the module, encompassing the details of the scenarios, the structure of discussions, and the corresponding subject topics. Relevant technical and non-technical artifacts are available upon request.

Besides finding the scenarios intriguing, student feedback emphasized the significance of employing a real-world project as a legacy product versus starting a new one. The open source provided a practical framework for applying design principles and patterns, which particularly resonated with students who had prior industry experience. The discussed method shows excellent potential in software design education, effectively preparing students for their future careers. However, the notable extra effort and resources required challenge its sustained implementation. The paper concludes with several successful practices and recommendations planned to guide the future iterations of this module while ensuring its continued relevance, engagement, and alignment with the changing needs of the software industry.

## REFERENCES

- [1] Barcelos, R. F., and Travassos, G. H. (2006). Evaluation Approaches for Software Architectural Documents: a Systematic Review. CIBSE, 433-446.
- [2] Begel, A., and Simon, B. (2008, March). Struggles of new college graduates in their first software development job. In Proceedings of the 39th SIGCSE technical symposium on Computer Science education (pp. 226-230).
- [3] Borque, P., and R. Fairley. "Guide to the software engineering body of Knowledge version 3.0." IEEE Computer Society Staff (2014).
- [4] Brechner, Eric. "Things they would not teach me in college: what Microsoft developers learn later." Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications. 2003.
- [5] Brown, S. (2015). Software Architecture for Developers—Visualise, Document and Explore Your Software Architecture. Victoria: Leanpub.
- [6] Cámará, Javier, et al. "On the assessment of generative AI in modeling tasks: an experience report with ChatGPT and UML." Software and Systems Modeling (2023): 1-13.
- [7] Codescene, 2024. <https://codescene.io/> 2024.
- [8] GitUML: UML visualizations for git repositories. <https://gituml.com/> 2023
- [9] Gonzalez-Huerta, J., Mollerí, J. S., Šablis, A., and Zabardast, E. (2020). Experiential Learning Approach for Software Engineering Courses at Higher Education Level.
- [10] Hu, Chenglie. "The nature of software design and its teaching: an exposition." ACM Inroads 4.2 (2013): 62-72.
- [11] Karre, Sai Anirudh, Lalit Mohan Sanagavarapu, and Y. Raghu Reddy. "Using a project-based approach to teach design patterns: An Experience Report." 14th Innovations in Software Engineering Conference (formerly India Software Engineering Conference). 2021.
- [12] Jia, Yanxia, and Yonglei Tao. "Teaching software design using a case study on model transformation." 2009 Sixth International Conference on Information Technology: New Generations. IEEE, 2009.

- [13] Murtagh, Jeanne L. "Teaching a real-world software design approach within an academic environment." 2000 Annual Conference. 2000.
- [14] Pinto, Gustavo Henrique Lima, et al. "Training software engineers using open-source software: the professors' perspective." 2017 IEEE 30th conference on software engineering education and training (CSEE&T). IEEE, 2017.
- [15] Mermaid. 2024. Mermaid Diagramming and charting tool. <https://mermaid.js.org/>.
- [16] Sonarqube, Sonar 2024. <http://www.sonarqube.org>
- [17] Maia, Marcelo. "Crossed Teams: Leveraging Student Interaction in Software Engineering Practice." Proceedings of the XXXVII Brazilian Symposium on Software Engineering. 2023.
- [18] Pedroni, M., Bay, T., Oriol, M., and Pedroni, A. (2007). Open source projects in programming courses. ACM SIGCSE Bulletin, 39(1), 454-458
- [19] PlantUML. 2022. PlantUML. <https://plantuml.com/>.
- [20] Pylint. 2024. Pylint 3.1.0-dev0 <https://github.com/PyCQA/pylint>. 2024.
- [21] Pyreverse. 2024. Pylint 3.1.0-dev0 documentation <https://pylint.readthedocs.io/en/latest/pyreverse.html>.
- [22] Ryan, B., Soria, A. M., Dreef, K., and van der Hoek, A. (2022, May). Reading to write code: an experience report of a reverse engineering and modeling course. In Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: Software Engineering Education and Training (pp. 223-234).
- [23] Souza, M., Moreira, R., and Figueiredo, E. (2019, September). Students' perception on the use of project-based learning in software engineering education. In Proceedings of the XXXIII Brazilian Symposium on Software Engineering (pp. 537-546).
- [24] Stähli, D., Sandahl, K., and Buffoni, L. (2022). An eco-system approach to project-based learning in software engineering education. IEEE Transactions on Education, 65(4), 514-523.
- [25] Marmorstein, R. (2011, June). Open source contribution as an effective software engineering class project. In Proceedings of the 16th annual Joint Conference on Innovation and Technology in Computer Science Education (pp. 268-272).
- [26] Smith, T. M., McCartney, R., Gokhale, S. S., & Kaczmarczyk, L. C. (2014, March). Selecting open-source software projects to teach software engineering. In Proceedings of the 45th ACM Technical Symposium on Computer Science Education (pp. 397-402).
- [27] Striuk, A. M., and Serhiy O. Semerikov. "Professional competencies of future software engineers in the software design: teaching techniques." Journal of Physics: Conference Series. Vol. 2288. No. 1. IOP Publishing, 2022.
- [28] Wiliam, D. (2011). What is assessment for learning? Studies in educational evaluation, 37(1), 3-14.
- [29] Van Deursen, Arie, et al. "A collaborative approach to teaching software architecture." Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education. 2017.

# “Think Before You Scrum” – An Experience Report on Daily Scrums to Aid Reflection

Matthew Minish

*Department of Computer Science  
and Software Engineering  
University of Canterbury  
Christchurch, New Zealand  
matthew.minish@pg.canterbury.ac.nz*

Matthias Galster

*Department of Computer Science  
and Software Engineering  
University of Canterbury  
Christchurch, New Zealand  
mgalster@ieee.org*

Fabian Gilson

*Department of Computer Science  
and Software Engineering  
University of Canterbury  
Christchurch, New Zealand  
fabian.gilson@canterbury.ac.nz*

**Abstract**—Project-based software engineering courses often incorporate industry practices, e.g., the “Daily Scrum” (i.e., daily stand-ups). While the main purpose of the Daily Scrum is to discuss progress and blockers, we can also utilize them as an educational tool to facilitate student engagement and reflection since reflection is a crucial aspect of experience-based learning. This paper seeks to enhance reflective practices in software engineering project courses through a *pre-meeting check-in* add-on to the Daily Scrum. We also aim to assess if this increases self-reflection and consequently impacts student engagement and performance. To this end, we adapted the Daily Scrum in a software engineering project course with a pre-meeting semi-guided self-reflection on a student’s work. This was supported by a bespoke online tool. Effects of the reflection were analyzed from the student perspective using surveys, application usage data, formal assessment data, and observations from teaching staff. From our experiences, we observed the pre-meeting check-in was beneficial in enhancing student self-reflectivity. Quantitative data, supported by qualitative evidence from student feedback, also suggested positive effects on student engagement and performance. Students who used pre-meeting check-ins reported increased self-reflection and showed greater levels of improvement in assessed performance. Based on our experiences we recommend incorporating regular individual self-reflection sessions in software engineering project courses.

**Index Terms**—Software engineering project courses, student reflection, stand-up meetings, daily Scrum

## I. INTRODUCTION

Software engineering project courses often use industry development methods and agile practices (e.g., Scrum), to facilitate collaborative and realistic learning [1], [2]. A key component of Scrum, the Daily Scrum (i.e., daily stand-ups), supports team communication, status updates, and problem-solving [3], [4]. While there have been efforts to explore the effects of the Daily Scrum on developers’ work in industry [4]–[7], we believe that with a slight modification, the Daily Scrum can also serve as an educational tool for facilitating individualized *student reflection*.

Reflection is a crucial aspect of experience-based learning, as it enables students to assess their progress, identify challenges, and improve their understanding [8], [9]. In this paper, we explore a modification of the Daily Scrum to include a *pre-meeting check-in*. Pre-meeting check-ins provide a unique opportunity for students to individually self-reflect on the

progress of their work since the last Daily Scrum, considering any experienced difficulties or lessons learned *before* the Daily Scrum. This is different to the purpose of the Daily Scrum, where the focus lies on inspecting the *team’s* progress [3], rather than providing space for individual reflection. Pre-meeting check-ins promote self-reflective behaviors in students by running them through a *short but regular, semi-guided reflection*. To facilitate the use of pre-meeting check-ins, we implemented a web-based tool integrated inside the project management tool used by students.

In this paper we present our findings on the use of pre-meeting check-ins in a project course, from both the student perspective: based on students’ perceptions and interactions with the intervention (i.e., the pre-meeting check-ins); and by investigating the impacts on assessed performance and engagement. This paper provides insights into our experiences with the intervention, in addition to offering recommendations for other software engineering educators looking to support student engagement and reflective practice.

In Section II we discuss background and related work. In Section III we detail the context of our work and present the design of the pre-meeting check-in intervention in Section IV. In Section V we introduce the data sources and collection method for this experience report. In Section VI we analyze the effects of pre-meeting check-ins on student engagement and reflection using these data sources, and finally, we provide recommendations for other software engineering educators and limitations in Section VII. We conclude in Section VIII.

## II. BACKGROUND

### A. Reflection in Agile Practices

The importance of reflection and reflective practices across all domains of education has been well established [10]–[12]. Project courses and experiential learning are commonly used teaching strategies in software engineering education [2], [13], [14] that implement an active learning process where students experience real-life scenarios and learn from their experiences through reflection [15]–[17]. Software engineering project courses are often designed around agile practices, particularly Scrum, which embed team-reflection into the development process for continuous improvement [18], [19].

Formal self-reflection and peer-feedback exercises are widely used in software engineering project courses [20], and predominantly align with Schön’s ‘reflection-on-action’ paradigm [10], a retrospective analysis where students reflect on their actions *after* they have occurred. However, moving towards a ‘reflection-in-action’ approach [10] will instead facilitate real-time problem solving and adaptation as challenges arise. To this end, Marques et al. showed enhanced learning in a project course with weekly ‘in-action’ reflections [21].

*1) The Daily Scrum as a Reflective Process:* The Daily Scrum is a Scrum event [3] in which project teams inspect their progress, and adapt their processes or commitments if necessary. This is an example of how reflection is used as an integral part of agile software development to guide how teams proceed. While the Daily Scrum is a reflective process, its usefulness for invoking *individual self-reflection* in an educational context may be limited in the following ways:

*a) Team Progress Reflection:* Individual reflection within the Daily Scrum is primarily to determine how individual efforts impact the team’s goals. While this reflection informs the team on its progress, it may not create adequate space for individuals, especially students, to introspect on the experience for learning. When the emphasis is on the team’s progress, students may miss out on reflecting upon their individual practices, which is crucial to learning in a project course.

*b) Managerial Reporting Reflection:* Daily Scrums run the risk of becoming ‘status-reporting meetings’ [7], where the focus shifts from a team’s inspection and adaptation towards the sprint goal, to individuals predominantly reporting their progress, often to a superior or manager. Not only does this pattern decrease opportunities for individual reflection and learning, but it has also been shown to have adverse effects [4], [6], [7], [22] on developers in industry teams, and should therefore be avoided. In an educational context where instructors are present in Daily Scrums to guide students, we must prevent a similar effect where students merely report status rather than engaging in collaborative or introspective reflections, reducing motivation and enthusiasm for the project, and a reduced learning experience overall.

*2) Retrospectives as a Reflective Process:* The Sprint Retrospective [3] in Scrum is an inherently reflective practice that can have a positive impact on a team’s process [23], [24]. Though they typically operate almost exclusively at a team-level [3], putting little focus on the kind of *individualised self-reflection* that is valuable for students in an educational context. In fact, retrospectives are often performed with a degree of anonymity to emphasise treating the team and its processes as a single entity and improve the overall psychological safety within the team [25].

## B. Related Work

In software engineering education, various strategies have aimed to foster reflective practices within project courses.

Burden and Steghöfer used an iterative–incremental process to facilitate continuous reflection for students [26]. Their approach hinged on the practical experiences gained through

activities such as the Lego Scrum simulation and the Elephant Carpaccio exercise, which engage students in agile methodologies and encourage iterative reflection. Our work builds upon their suggested future work for the incorporation of the Daily Scrum, by including a novel reflective intervention.

Marques et al. explored ‘Reflexive Weekly Monitoring’ in a project course [21], where teams would meet weekly with an external ‘monitor’. While the monitor was not responsible for a team’s activities, they would meet weekly to guide the team through a reflective process and offer advice or warnings from their own diagnosis of the team’s performance. The paper reported that this process was effective in enhancing student learning, as teams were more effective and coordinated, and experienced a higher sense of team belonging and satisfaction. Our research differs from theirs by focusing purely on the individual, and having no requirement for external staff or additional resourcing.

Babb et al. looks specifically at how Schön’s concept of ‘reflection-in-action’ can be embedded into agile development [27] in industrial settings. In their work, the Daily Scrum is highlighted as an occasion for recalling recent ‘reflection-in-action’ experiences, serving as a space to discuss and build upon individual and team reflections to foster ongoing learning and improvement. The authors offer the recommendation for agile practitioners to consciously harness standard agile practices to implicitly capture reflection-in-action, thereby unearthing learning opportunities that are often overlooked in daily routines. Our work builds on these recommendations, but is novel as we apply our intervention in an educational rather than industrial context, and have created a bespoke software tool more suited to the asynchronous nature of student work.

## III. TEACHING CONTEXT

### A. Software Engineering Project Course

*1) Overall Context:* We applied pre-meeting check-ins in a third-year software engineering project course at the University of Canterbury, Christchurch, New Zealand. Students in the course have a basic understanding of computer science fundamentals, having taken courses covering: introduction to software engineering processes, practices and techniques; object-oriented design and programming; programming in Python and Java; relational databases and SQL; formal languages and compilers; algorithms; and, a mixture of calculus and discrete mathematics. Many students also have experience from at least one summer internship.

The course is a two semester (24 weeks) project (not a capstone project) in which there are no lectures, but instead students work in teams following an adapted Scrum method framework to create complex applications (e.g., travel planning, social network). In 2022 there were eight teams, ranging from 6-8 members. The expected workload is roughly 15 hours per student, per week, including development work, Scrum events, and academic requirements. The project consists of seven sprints spread across both semesters. Each sprint is approximately three weeks long (ignoring term breaks when students are not required to work). The final sprint ends in

week 21, allowing for post-project interviews, end-of-year surveys, and a project showcase. Students are assessed both on their technical skills (e.g., process compliance, architecting, coding, testing), and inter-personal skills (e.g., pair programming, dedication, accountability, behavior in team).

2) *Implementation of Scrum:* All teams in the course start working towards the same product backlog, and all share a product owner (PO), who is a full-time academic teaching the course. Each team has a ‘Scrum Master’, being someone who has previously taken the course – typically a student in their final year or a PhD / Masters student. In addition to those roles specified in Scrum, the course also includes Senior Tutors that oversee the Scrum Masters, a Chief Technical Officer, and Quality Assurance Officer who are senior members of the teaching team. Teams perform all Scrum Events described in the Scrum Guide [3].

3) *The Daily Scrum:* During the Sprint Work, teams are required to have at least two Daily Scrums per week rather than daily. Daily Scrums occur face-to-face, with all team members standing around a Scrum Board and typically answering the following questions: What have I accomplished since the last Daily Scrum? What will I accomplish before the next Daily Scrum? Are there any impediments in my way? The Daily Scrum is strictly time-boxed to 15 minutes. Students are expected to come *prepared* to the Daily Scrum having reflected on the state of their tasks and updated the project management tool (Section III-B). Scrum Masters or other senior teaching staff may be present at Daily Scrums to guide students.

4) *Reflection Opportunities in the Course:* Before the pre-meeting check-ins intervention, we formally enacted student reflection in two ways: Sprint Retrospectives – performed in all but the final sprint of the project; and, self-reflection and peer-feedback forms required from students after each sprint.

The self-reflection and peer-feedback forms are performed individually, and are kept confidential between a student and the teaching team. These activities aim to create a personal reflection space adhering to Schön’s ‘reflection-on-action’ paradigm where students can candidly assess their own, and their peers’, performance during the sprint. The forms consist of a combination of: ‘Likert’ scale questions where students rate their peers’, or their own, performance across various dimensions; and, some free-text questions for more qualitative insights. The self-reflection and peer-feedback forms help to gauge individual and team performance regarding: contributions to implementation and testing; communication with the team; and, overall effectiveness. They encourage students to critically identify areas for personal skill enhancement in themselves and their team members.

Students will typically spend between 1 and 3 hours (total) filling in the self-reflection and peer-feedback forms each sprint. These forms are assessed on a ‘traffic light’ scale: a ‘Green’ grade requiring both insightful comments and *Specific, Measurable, Achievable, Relevant, and Time-bound* (SMART) steps for improvement; a ‘Yellow’ grade being awarded for insightful comments that lack SMART steps for improvement; and, a ‘Red’ grade when neither of these criteria are met.

## B. Project Management Tool

For the purpose of this project course, we introduced a tailored software project management tool, aptly named ‘ScrumBoard’. The ScrumBoard was designed for the Scrum framework, while being tailor-made for students to reduce cognitive load by avoiding unnecessary and unused functionality present in more comprehensive industry tools such as Jira. The tool allows students to: create and organise a backlog of user stories; break-down user stories into time-estimated tasks; track the status of tasks each sprint with a drag-and-drop sprint board; log time spent working on tasks or attending Scrum Events; visualise team progress with burn-down charts and cumulative flow diagrams; and, view aggregated statistics about their own contributions to the project. Its interface and functionality were made to closely mirror industry-standard tools like Jira. Furthermore, the tool meticulously logs all user actions, including a complete history of all data in the system, allowing researchers and assessors a reliable and replayable timeline of all usage events.

Finally, the tool provides a Daily Scrum scheduling feature, and it was enhanced to support the intervention with a pre-meeting check-in feature to receive students’ reflection.

## IV. PRE-MEETING CHECK-INS

As discussed in Section II, the degree of *individual* reflection to support student learning in frameworks like Scrum is low, and largely contained to a few short bursts during a course. Therefore, we designed an intervention which would integrate self-reflection into a student’s everyday process.

### A. Intervention Design

When devising the intervention we specified several guiding principles grouped into three categories to represent the multifaceted approach in the teaching context.

1) *Educational Pedagogy:* Any intervention in an educational context with students should be firmly grounded in educational pedagogy. The following requirements are informed by the well established importance of reflection in experiential and project-based learning [15], [16], [28].

**P-Edu1** The intervention should increase the frequency of *individual* self-reflection.

**P-Edu2** The intervention should support students in meaningful reflection, inciting critical introspection rather than managerial-style status-reporting.

2) *Relevancy to Software Engineering:* Our goal is not simply to make software engineering students reflect; rather, we are aiming to make student software engineers *more reflective* as a whole (i.e., reflection as a professional skill). By integrating reflection more closely with a typical software engineering workflow we increase the likelihood of students retaining this reflectivity as they move into the software industry [10], [17].

**P-SE1** The intervention must be firmly rooted in the software engineering domain.

**P-SE2** The intervention should feel to students like a cohesive process of agile software development.

3) *Practicality*: To be successfully applied to a project course, the intervention should neither disrupt students' workflows [29], nor increase course teaching staff's workload [30].

**P-Pr1** The intervention will cause minimal disruption to a student's typical way of working.

**P-Pr2** The intervention must not take up any additional time from course teaching staff.

### B. Intervention Implementation

Based on these principles, we propose an intervention, the 'Pre-meeting Check-in'. This check-in takes the form of a short process wherein a student will, *before* each Daily Scrum, write a reflection on the progress of their work since the last Daily Scrum in the project management tool, i.e., select the status of each task they worked on, and optionally, a reflection about the progress of each task. This reflection is performed through an interface in the project management tool and should take no more than a few minutes to minimise overhead.

Using the Daily Scrum as the intervention's entry-point shifts the frequency of reflection from once per sprint to (in our context) at least twice per week, thus satisfying **P-Edu1**. Similarly, merging the intervention with an existing Scrum Event which already includes some degree of preparation by developers supports cohesion with the typical developer workflow (**P-SE1**, **P-SE2**, and **P-Pr1**). To guide students towards critical introspection and away from basic status-reporting (**P-Edu2**), the check-in asks a student to input the following information about each of their current tasks:

1) **Required** - The current state of the task (chosen from the following pre-defined options to discourage students from spending time on reporting status):

- *This task is complete, all changes have been merged.*
- *This task is complete but not merged, because it is yet to be reviewed.*
- *This task is not complete, I will keep working on it.*
- *This task is not complete, I will keep working on it but with someone to help.*
- *This task is not complete, it is too big and should be broken down into smaller tasks.*
- *This task is not complete, I will give it to someone else.*

2) **Optional** - Free-text space for reflection on their experiences working on the task.

To further nudge students away from status-reporting and towards meaningful self-reflection, the free-text space for reflection had the following placeholder text: "*Comment about your process when working on this task, for example: 'I should have started this task earlier because...', 'Next time I am struggling on a task like this I will...', 'While working on this task I tried...'* ". A tool-tip element was also shown nearby advising students to use these comments to reflect on their experience, rather than to report the status of their tasks.

It was also made very clear to students that no member of the teaching staff was able to access check-in responses during the project, and they would therefore not be used in any way for assessment during the course. This would ideally

help students to feel comfortable with candidly and critically analysing their own work as they would not fear being marked down for discussing mistakes they had made, but it also meant that there was no opportunity for teaching staff to help students improve the quality of reflection in the check-ins. This is discussed more as a limitation in Section VII-C4.

Finally, as the entire pre-meeting check-in process is managed within the project management tool, we introduce no additional work for course teaching staff, thereby satisfying **P-Pr2**. Similarly, we adhere to **P-Pr1** by having check-ins implemented as a part of the project management tool – which students are already interacting with multiple times per day.

### V. SOURCE OF EXPERIENCE

We used the intervention in the 2022 instance of the project course and focused on the following questions:

**Q1** How can we increase the frequency of student reflection in software engineering project courses?

**Q2** Does increasing the frequency of self-reflection in a software engineering project course with pre-meeting check-ins increase student performance?

During the sixth (of seven) sprints, we divided the class into two groups: an 'observational' group which used pre-meeting check-ins; and, a 'control' group which did not use pre-meeting check-ins. Initially, each group consisted of four teams, with the total number of students in each group being 25 and 26 for control and observational respectively. Teams were allocated to either of the groups so that the two groups were as similar to each other as possible with respect to: average time spent on the project per week; GPA; and, assessed performance in the course to date.

Our insights are derived from the following data sources:

- 1) Analysis of student responses to the pre-meeting check-ins by teaching staff after completion of the project, to determine if the check-ins were used as intended and to look for any noticeable trends in student reflections.
- 2) A survey conducted at the conclusion of the observational period, when only half of the class had used the pre-meeting check-ins. The survey aimed to gather student opinions on both their typical reflection practices overall, and with regards to check-ins (where applicable).
- 3) Elements of the course's formal assessment data, to explore any noticeable differences in performance between students who engaged with the check-ins and those who did not. This includes data from the project management tool, which provides quantitative insights into student work habits via student-created work-logs which document instances of work completed within the course.

The collection and usage of data have been approved by our institution's Human Research Ethics Committee.

### VI. FINDINGS

#### A. Post-hoc Groupings

Once we had begun to analyse the data generated during the observational period, we observed that students could be

more accurately grouped into *three* distinct groups based on their interactions with the intervention (rather than one control and one treatment group), see Sections VI-A1 and VI-A2.

1) *Changes to Control Group:* One team of students who were originally assigned to the control group never performed a single pre-meeting check-in, nor did they ever even access the intervention via the project management tool (as we could verify from the system usage logs). All but one of the students from this team completed the post-observation survey, and each of these students in some way stated that they never used the tool because they were not aware that the intervention (i.e., pre-meeting check-in) even existed. We therefore interpret the lack of engagement from this team as a miscommunication with the researchers, rather than an instance of students *choosing* not to use the intervention. For these reasons we have re-categorised post-hoc these students who did not use the tool as being part of the control group. This is discussed further as a limitation in Section VII-C1.

2) *Changes to Observational Group:* Regarding the remaining observational students, we observed from their usage of the intervention that students used it in one of two ways.

A majority of these students (15/20) used the tool as expected – to critically reflect on their experiences, but a smaller though not insignificant group (5/20) of the students used it exclusively to report the status of their tasks, e.g., whether the task was completed, what work (if any) remained, despite the researchers emphasising the reflective nature of the exercise. Such check-ins did not involve any introspection, but were a factual description of project artefacts. Therefore, our analysis distinguishes between the “status-reporting” usage of the intervention, and the intended “reflective” usage. It should be noted that almost all check-ins, i.e., including those where students used the tool as expected, included some description of the task’s status, similar to the “status-reporting” behaviours observed in industry software teams by Stray et al. [7].

3) *Final Group Definitions:* The groups used for the majority of this paper’s results and discussion, are hereby defined:

- 1) **Control group** - All students who did not perform any check-ins during the observational period, either because the check-ins were not available to them, or because the check-ins were available, but were not known about.
- 2) **Group under observation** - Separated into two sub-groups:
  - a) **Status-reporting group** – The students from the observational group that only used the tool to report their tasks’ statuses, not using it in any way for critical self-reflection.
  - b) **Reflective group** – The students from the observational group that had used the tool as intended, for reflecting on their individual experience since the prior Daily Scrum.

4) *Demographic Information:* The need to re-define post-hoc the student groups has lead to a lower degree of similarity between groups (Section V) used in our final analysis compared to the initial groups. These differences are shown in

TABLE I  
GRADING SCALE USED IN THE COURSE. ALL GRADES EXCEPT D AND E ARE PASSING GRADES.

Grade	GPA Value	Marks
A+	9	90-100
A	8	85-89.9
A-	7	80-84.9
B+	6	75-79.9
B	5	70-74.9
B-	4	65-69.9
C+	3	60-64.9
C	2	55-59.9
C-	1	50-54.9
D	0	40-49.9
E	-1	0-39.9

Table II. The metrics included in this table are:

- # – the total number of students included in the group;
- *Weekly hours* – the mean number of hours logged against the project per week, not including the time taken in course ‘overhead’ sessions or formal Scrum events;
- *Average GPA* – mean GPA of students in the group at the start of the course (see Table I);
- *Average Sprint Grade* – mean of sprint grades awarded to students for sprints 2-5 (i.e., once they had joined with their full teams until just before the commencement of the observational period).

The post-hoc categorisation of students impacted the similarity between the different groups, most notably in terms of the averages for GPA and previous Sprint Grades.

TABLE II  
BASIC PERFORMANCE METRICS BY STUDENT GROUPING. STANDARD DEVIATION IN PARENTHESES.

	Group	#	Weekly hours	Average GPA	Average sprint grade
Pre	Control	25	7.48 (2.09)	5.89 (1.38)	58.0% (9.2%)
	Observation	26	6.80 (1.18)	6.36 (1.59)	61.6% (8.9%)
Post	Control	31	7.22 (1.97)	6.01 (1.43)	58.4% (13.8%)
	StatusReport	5	7.05 (0.74)	6.12 (1.45)	60.0% (12.0%)
	Reflective	15	6.94 (1.30)	6.36 (1.78)	62.8% (12.6%)

#### B. Pre-meeting Check-in Data

During the observational period, 20 students performed a total of 385 check-ins. 173 check-ins included free-text entries by the students. Through *expert discussions*, two researchers manually looked at all check-ins and categorised 69 of them as ‘reflective’ because they contained personal reflection beyond just describing the status of the task. Reflective check-ins averaged 38 words, and non-reflective averaged 17 words. Status-only check-ins typically included descriptions like ‘Just needs to be merged in’ or ‘Almost finished, just need to [...]’. In contrast, reflective check-ins’ text varied greatly in form and substance; examples of such check-ins are given below.

When categorising check-ins, we noticed that most reflections could be broadly grouped within one of three categories: problem-solving processes, teamwork, and self-awareness.

1) *Reflection on Problem Solving*: This theme is related to reflections on the technical challenges students faced, often regarding their interactions with demanding tasks or bugs.

For example, one student reflected: *'It took me a long while to complete this task because I poorly understood the output of what I was testing. In future, I could try to examine things more closely, or alternatively push for better documentation so that it's clearer to people reading.'* This reflection reveals the student's recognition of their learning opportunity – not just at a personal level, but also for the wider team to improve their documentation. This identification of steps for personal and team improvement highlights the potential learning value of check-ins for improving student problem-solving skills.

2) *Reflection on Teamwork*: Teamwork is a core learning outcome for software engineering. Reflective check-ins revealed diverse learning opportunities for student teamwork skills, from confidence building and speaking up, to active listening, empathy, and better understanding of body language.

One student reflected: *'Pair programming was useful for this task, but I still need to improve my ability to read body language for when someone wants to take the keyboard, and when they want to let someone else do the typing.'* This reflection shows that the student values pair programming and identifies a specific area for personal development – interpreting non-verbal cues during collaboration.

Another student shared: *'I worked on this with [team member X], doing online pair programming. Online pair programming is not as bad as I had expected, and I regret not doing online pair programming with [other team members] earlier in the year.'* This reflection reveals a change in the student's perception of online pair programming. It suggests a shift in mindset, a valuable aspect of learning, underscoring the benefit of reflective check-ins in prompting students to reassess their preconceptions and adapt their practices.

3) *Self-awareness*: Self-awareness is another significant outcome in software engineering project courses. The reflective check-ins frequently highlighted a change in students' changing perception of their skills and preferred working practices, often following specific struggles or triumphs.

For instance, one student shared: *'This is a relatively fulfilling task for me. Although it took a long time and only changed three lines of code in the end, the process of finding the source of the error was very rewarding. Front-end debugging will be easier in the future. I'm glad I was able to do this task independently without asking for help.'* This reflection describes an instance of a student that had traditionally avoided difficult tasks – or had often needed help to complete them. Despite the complexity and small number of code changes, they gained confidence in their ability to tackle challenging tasks, reflecting a positive learning outcome.

### C. Survey Responses

A survey was distributed to all students on the final day (Friday) of Sprint 6 once all formal events for the sprint had been completed, and was open until the following Monday. It was not compulsory, and was answered by 46 out of 51

students (90%): 27 out of 31 (87%) control students; 4 out of 5 (80%) status-reporting students; and all 15 (100%) reflective students. The median time to finish the survey was 9.6 minutes, with the shortest response taking only 2.7 minutes, and the longest response 28.5 minutes. The survey included questions about: feelings towards the Daily Scrum; individual student reflection during sprints; and for the initial observational group, evaluation of the pre-meeting check-ins intervention. For simplicity, and in part due to the very small size of the status-reporting group, we have combined the responses of the status-reporting and reflective students in this paper.

1) *Incidence of Reflection*: Figure 1 shows agreements with the statement: *'During [the observational period] I spent more time reflecting on my current tasks before the Daily Scrum than I had previously'*. Assigning a numerical value of -2 to 'strongly disagree' and 2 to 'strongly agree', students in the post-hoc control had a median agreement of 0 (neutral). Conversely, the median agreement score of students in the status-reporting and reflective groups was 1 (agree).

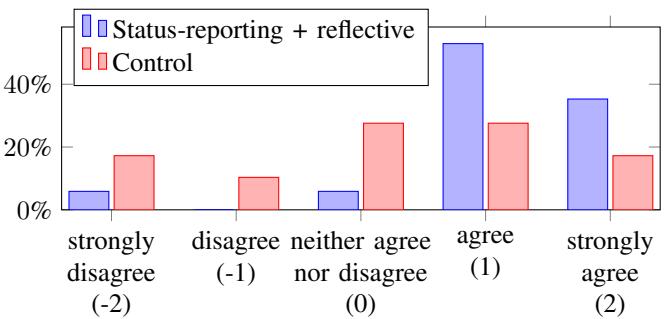


Fig. 1. *'During [the observation period] I spent more time reflecting on my current tasks before the Daily Scrum than I had previously'*

Two survey questions identified what effect on reflection was brought about specifically by the intervention. Figure 2

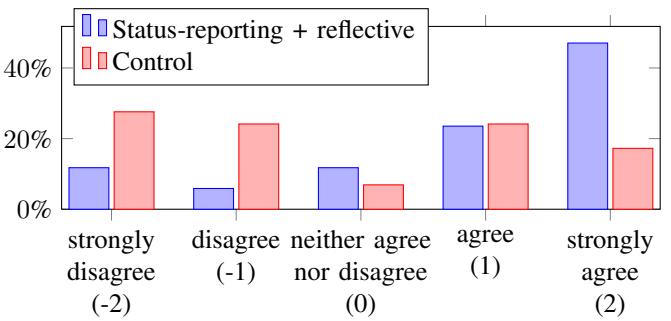


Fig. 2. *'During [the observational period], seeing the upcoming Daily Scrum notification prompted me to reflect on my current tasks'*

shows that the post-hoc control students had a much lower agreement with the first statement than their status-reporting and reflective counterparts, having a median agreement of -1 (disagree). Comparatively, the median agreement rating for status-reporting and reflective students was 1 (agree). Figure 3 shows a high level of agreement by the reflective and status-

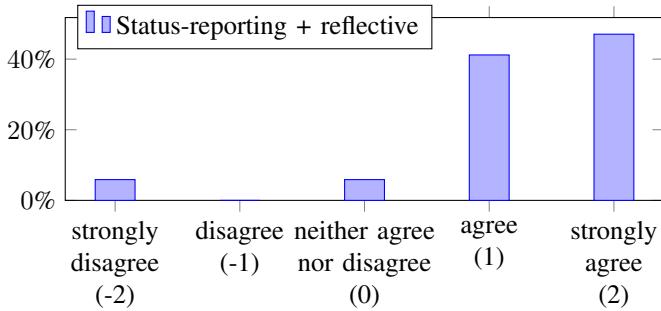


Fig. 3. ‘During [the observational period], performing the pre-meeting check-ins caused me to reflect on my current tasks’

reporting students with a median agreement score of 1 (agree), with all but two students agreeing with the statement.

The largely positive agreement by the status-reporting and reflective students to these two statements suggests that pre-meeting check-ins cause students to reflect on their current tasks, and even the reminder of an upcoming Daily Scrum (and by extension, check-in) may prompt some degree of reflection.

#### D. Formal Assessment

In this section we compare the differences in formal assessment between the three post-hoc groups. When comparing grades between sprints and across groups, it is important to note that we expect to see an increase in average grades over time as students continue to practice and improve. Therefore, the ‘control’ group can be used as a baseline for comparison of the performance of students who used the pre-meeting check-ins intervention. As we see in the following sections, while the average performance of all groups increased, the ‘control’ group showed the lowest level of increase of all three groups.

Note that the final sprint of the course, Sprint 7, is not discussed in this section as it does not include many of the assessment artefacts and formal events used in other sprints. In the final sprint students do not submit a self-reflection, nor do teams perform Sprint Reviews or Retrospectives. Additionally, many students treat the final sprint as a time for final ‘polishing and fixing’, where typical workflows are largely disrupted.

1) *End of Sprint Self-Reflection Assessment:* Students filled out a ‘Self-Reflection’ at the end of every sprint which was assigned a ‘Traffic light’ grade by teaching staff (see Section III-A4). Here, we compare the assessed performance of the three post-hoc groups before and after the observational period. Assessment is shared across all teaching staff, with the course coordinator reviewing all marks to ensure consistency.

The traffic light scores of ‘Red’, ‘Yellow’, and ‘Green’ were assigned the values 0, 1, and 2 respectively. Thus, we calculated a mean grade for each group for two time-periods: the pre-observation period (i.e. Sprints 2-5, after students had been placed in final teams and before the intervention had been introduced); and, the observational period – Sprint 6.

Table III shows the post-hoc ‘control’ group had a slightly lower average Self-Reflection score in the pre-observational

TABLE III  
ASSESSED QUALITY OF END-OF-SPRINT SELF-REFLECTIONS BY STUDENT GROUP. STANDARD DEVIATION IN PARENTHESES.

	Pre-observation	Post-observation	Diff	Nml Gain
Control	1.21 (0.56)	1.29 (0.52)	0.08	0.10
SttsRppt	1.28 (0.60)	1.4 (0.49)	0.12	0.17
Reflective	1.29 (0.62)	1.6 (0.49)	0.31	0.44

TABLE IV  
ASSESSED SPRINT GRADES BY STUDENT GROUP USING AVERAGE OF PRE-OBSERVATIONAL PERIOD (SPRINTS 2-5). STANDARD DEVIATION IN PARENTHESES.

	Pre-observation	Post-observation	Diff	Nml Gain
Control	58.4% (13.8%)	68.8% (8.4%)	10.4%	25.0%
SttsRppt	60.0% (12.1%)	77.0% (7.1%)	17.0%	42.5%
Reflective	62.8% (12.6%)	84.6% (10.2%)	21.8%	58.6%

period than their ‘status-reporting’ and ‘reflective’ classmates with a mean score of 1.21 versus 1.28 and 1.29 respectively.

When considering the assessed Self-Reflection scores during the observational period, we observe a much larger difference between the groups. Looking at normalised gain “g” (a measure that quantifies the relative improvement from pre-test scores,  $\langle \text{pre} \rangle$ , to post-test scores,  $\langle \text{post} \rangle$ , calculated as  $g = \frac{\langle \text{post} \rangle - \langle \text{pre} \rangle}{100 - \langle \text{pre} \rangle}$  [31]), we saw a slight increase in the mean ‘control’ score, for a normalised gain of 0.10. The ‘status-reporting’ group saw a slightly larger increase with a normalised gain of 0.17. Meanwhile, the ‘reflective’ group saw a much bigger increase, with a normalised gain of 0.48.

TABLE V  
ASSESSED SPRINT GRADES BY STUDENT GROUP FOR SPRINTS 5 AND 6. STANDARD DEVIATION IN PARENTHESES.

	Sprint 5	Sprint 6	Diff	Nml Gain
Control	63.8% (12.5%)	68.8% (8.4%)	5.0%	13.8%
SttsRppt	70.0% (9.1%)	77.0% (7.1%)	7.0%	23.3%
Reflective	70.0% (8.9%)	84.6% (10.2%)	14.6%	48.7%

2) *End of Sprint Overall Assessment:* Each student is awarded an indicative grade in every sprint based on their performance. These grades are assigned by the teaching team as a percentage grade, which is then mapped to a letter grade range before being shown to students (see Table I for the mapping to letters). Table IV shows the average grade achieved by each group during the pre-observational period (Sprints 2-5), the average grade achieved during the observational period (Sprint 6), and the normalised gain of each student group. We use an average grade from this range as sprint grades will commonly fluctuate up and down throughout the year. Table V shows a direct comparison of student grades, and the calculated normalised gain, when considering only the observational period (Sprint 6), and prior sprint.

Tables IV and V show that – as expected – the average performance across all student groups increased over time, however the size of increase across the groups was certainly not uniform. The ‘control’ group improved from their pre-observational average with an average increase of exactly two letter grades, moving from 58.4% (‘C’ grade), to 68.8%

(‘B-’ grade). This was an increase of one letter grade from their average Sprint 5 performance of 63.8% (‘C+’ grade). Comparatively, the ‘status-reporting’ and ‘reflective’ student groups displayed a noticeably higher improvement from both their pre-observational (Table IV) and their Sprint 5 averages (Table V). Per Q2, this indicates that increased reflection can cause an observable improvement in student performance.

The ‘status-reporting’ group improved just over 3 letter grades from their pre-observational average of 60.0% (‘C+’ grade) to 77.0% (‘B+’ grade). This was an increase of one letter grade from their average Sprint 5 performance of 70.0% (‘B’ grade). The ‘reflective’ student group showed the largest improvement on both tables, improving upon their pre-observational average of 62.8% (‘C+’ grade) to 84.6% (high ‘A-’ grade) – an increase of just over 4 letter-grades. This represented an increase of just under three letter grades from their average Sprint 5 performance of 70.0% (‘B-’ grade).

Figure 4 shows that the change in performance of the ‘reflective’ student group (14.6%) during the observational period was the biggest sprint-on-sprint improvement observed for any group during the entire year.

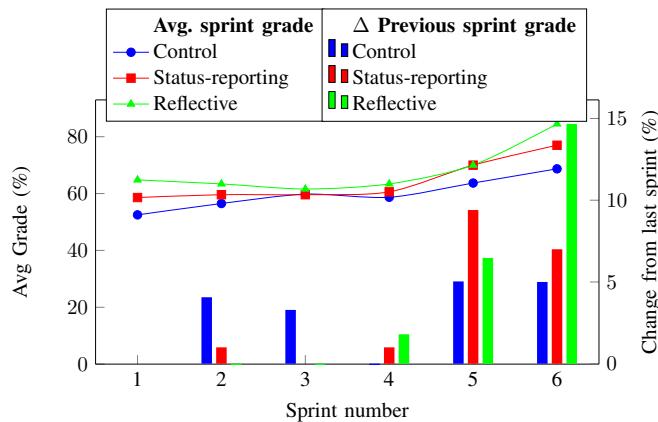


Fig. 4. Average grade per sprint per control group (line chart), and change in average grade since last sprint (bar chart)

## VII. DISCUSSION

We first consider what our insights mean in terms of our initial questions (see Section V, and then present our recommendations.

### A. Intervention Objectives

1) **Q1:** *How can we increase the frequency of student reflection in software engineering project courses?*: Section VI-B gives examples where students used the check-ins to self-reflect on their problem solving or teamwork processes, alongside actively engaging in self-aware reflections.

Furthermore, the survey responses detailed in Section VI-C show that students who used the check-ins reported an increase in reflection both when performing the check-ins, and in the wider sprint overall. Specifically, Figure 1 shows that the students who interacted with the check-ins agreed that they

spent more time reflecting during the observational period than they had previously, this agreement being much stronger than that of their peers who did not interact with the intervention. Figure 2 shows that for students who used the intervention, merely seeing the reminder for an upcoming Daily Scrum may have prompted reflection. Finally, Figure 3 shows a very strong agreement by students that the use of the intervention directly caused them to reflect more on their current tasks.

**These results indicate that the proposed pre-meeting check-ins may represent a valid approach for promoting student reflection** in software engineering project courses, though there were one issue we noticed around how students interacted with the check-ins that should be considered. The pre-meeting check-ins included to a high degree of ‘status-reporting’ by students, rather than reflection. Such behaviors have been identified as detrimental in industry teams [4], [7], are not representative of the critical self-reflection we are trying to encourage in students, and cause unnecessary double-up with the first question of the Daily Scrum: “What have I accomplished since the last Daily Scrum?”.

2) **Q2:** *Does increasing the frequency of self-reflection in a software engineering project course with pre-meeting check-ins increase student performance?*: By using formal assessment items from a project course, we were able to compare the performance of students both to themselves over time, and also against other groups based on their usage of the pre-meeting check-ins. We expected that all student groups would, on average, improve over time as students continue to practice and learn from their experience in the course, and indeed we observed such an improvement across all student groups. Section VI-D shows that while all student groups improved, the student groups who used the pre-meeting check-ins on average improved more than those who did not. Additionally, the student group that used the check-ins for self-reflection also showed a higher average improvement in assessed performance than the students that only used it to report status of tasks.

### B. Lessons Learned and Recommendations

Based on our reported experiences, we list the following lessons learned as recommendations to other educators.

1) **Recommendation 1: Include individual self-reflection as a regular part of the agile process in project courses:** Agile software development and frameworks like Scrum already place a high importance on continuous reflection, but this reflection is typically not the kind of individualised self-reflection that benefits student learning in experiential scenarios like project courses. We reported on a study involving the inclusion of a formal reflective practice as an *addition* to the Daily Scrum. We observed an increase in learning of the students who used this intervention as a reflective tool.

The difference in learning gain between the groups shows that while such an intervention is not required for some learning to occur – as the control student group still showed an improvement in learning, the degree of learning brought about by reflection may be significantly increased by integrating

a regular self-reflective process into the typical workflow of students in a software engineering project course.

*2) Recommendation 2: Make student reflections a ‘safe space’:* We observed that students tended to focus on detailing the current status of their work during the check-ins rather than discussing their own processes and experiences – despite the tool’s instructions specifically stating that students should comment on their own process, and example prompts being given that emphasised self-reflection. Similar status-reporting behaviors have been observed in industry teams and linked to self-justification [4], [7]. Therefore it seems likely that students may similarly feel a need to justify themselves by reporting the status of their work done during reflections, rather than critically analysing their own work to find ways to improve.

For this reason, we recommend that any reflections that students perform within a project course should be treated as a ‘safe space’, where students can discuss their failures without fear of being assessed negatively because of them, and without thinking that they need to justify themselves by detailing the work that they have performed. In this same way, we recommend that where student reflections are somehow assessed, that the assessment should be based only on the quality of the reflection itself, so that the student may receive guidance on how to *better* reflect, while still feeling comfortable to openly discuss their experiences.

*3) Recommendation 3: Students should be given guidance on how to reflect:* While the status-reporting behaviors observed in check-ins might be explained by students feeling the need to justify their actions to avoid negatively impacting their assessment in the course, another cause may be that students are not properly equipped to perform good self-reflections.

Therefore, following on from recommendation 2 above, we recommend that for educational practices that are intended to promote self-reflection, guidance should be given to students for how to concretely enact these practices in a self-reflective manner. Particularly, students should be taught that a good reflection will focus on their own actions, experiences, and emotions [8] – rather than discussing the status of the work, except where it helps to contextualise the student’s actions, experiences, or emotions. We recommend including this guidance alongside other continuous assessment feedback, for example on a per-sprint basis. Any such feedback should explicitly state that only the quality of the reflections has been considered, in keeping with Section VII-B2.

### C. Limitations

As an experience report, the results and recommendations given in this paper are influenced by the following limitations.

*1) Class Size:* While the intervention proposed in this paper was successfully applied to a real education context as described in Section III-A, the total number of students was only 51. This was compounded by our choice to separate the class into a control and observational group, and then further exacerbated by 6 of the 26 observational students never interacting with the intervention whatsoever, leading to their post-hoc inclusion in the control group. On the other hand,

we did not aim for statistical generalization, but aimed at: (1) introducing a new intervention to facilitate reflection; (2) demonstrating its feasibility to contribute to student learning by analyzing detailed data from the project management tool and other sources; and (3) providing recommendations for others who would like to adopt the approach based on our experience using the intervention.

*2) Selection Bias and Disparities in GPA:* We cannot be certain that the only difference between our groups was the use of the intervention, particularly given the need to re-categorise these groups post-hoc, and disparities in GPAs between both groups. In fact, Table II shows that even before the intervention was applied, the ‘reflective’ student group had a higher average GPA and sprint grade than the ‘status-reporting’ group, who likewise had a higher average GPA and sprint grade than the ‘control’ group. This may indicate the possibility of selection bias, where we can not distinguish between the notions that pre-meeting check-ins may have caused students to be more reflective and therefore perform better in the course, and that students who were already more reflective and likely to perform more highly in the course would in turn be more likely to interact with the check-ins.

*3) Timing of Intervention:* There were 7 total sprints throughout the course, and the intervention was not applied until the 6th of these sprints, approximately two thirds of the way into the course. By this point in the course, students had already learned a great deal about what was required for success in the course, and so many were already on an ‘upwards trajectory’. This is clearly shown in Figure 4, where we saw a significant increase in the assessed performance of students, before the intervention had been applied.

*4) Unintended Status-reporting in Intervention:* Students interacted with the intervention differently than how we intended, including a large degree of status-reporting in their check-ins rather than focusing on self-reflection. In fact, some students exclusively used the tool for status-reporting without any self-reflection, leading partly to the post-hoc separation of groups as detailed in Section VI-A. While the emergence of this unexpected ‘status-reporting’ group has highlighted similarities to the status reporting behaviors commonly spotted in industry [4], [7], and consequently informed our recommendation in Section VII-B2, it has limited our ability to assess the effectiveness of our intervention as a tool for self-reflection.

The observed status-reporting also makes us question our decision to keep the check-ins private from teaching staff. This decision was made as a conscious effort to help prevent status-reporting from students, as we suspected they may not be as willingly to critically discuss their mistakes or perceived failures if they thought it could be read by teaching staff and possibly hurt their assessment. However, in retrospect – and aligning with our Recommendation 3 in Section VII-B3 – we consider that the root cause of the status-reporting may rather have been that some students simply lacked the proper skills to reflect meaningfully, an issue that could have been alleviated by teaching staff viewing check-in responses in order to offer students guidance on how to improve their reflection.

### VIII. CONCLUSION

We report our experience with pre-meeting check-ins to increase student self-reflection in software engineering project courses. The check-ins were created in accordance with guidelines grounded in educational pedagogy and literature, and have been successfully applied to a real educational setting.

We assessed the check-ins from a student perspective through a survey and by investigating how students interacted with the check-ins, including a number of example check-ins from students that illustrated the potential benefits to student self-reflectivity across a number of important topics in project courses. We also used assessment data from the course and application usage data gathered from the project management tool students used, to quantitatively analyse the intervention's effect on student performance and engagement.

We developed a number of recommendations for educators seeking to improve student learning in software engineering project courses regarding how to encourage student self-reflection. Our future work includes the following:

**Include guidance on correct usage of intervention:** Future applications of the pre-meeting check-ins should be accompanied by an explanation to students of the differences between status-reporting and reflection, encouraging the use of the check-ins for the latter. This explanation should accompany the initial introduction of check-ins to students, and should also be included in the technical implementation of the project management tool, for example as a short 'tutorial'.

**Make intervention available for all students equally:** The pre-meeting check-ins intervention should be made available to all students equally rather than having a control group of students who could not access the tool. This is to both ensure that no students are potentially disadvantaged by not having access to the tool as they may have been in this iteration of the study, but also to increase the potential sample size of students interacting with the intervention.

**Introduce earlier in the course:** The pre-meeting check-ins intervention should be made available much earlier in the course's life-cycle to encourage students to reflect while still having as much time as possible to adapt their future practice.

### REFERENCES

- [1] D. A. Umphress, T. D. Hendrix, and J. H. Cross, "Software process in the classroom: The capstone project experience," *IEEE software*, vol. 19, no. 5, pp. 78–81, 2002.
- [2] V. Mahnic, "A capstone course on agile software development using scrum," *IEEE Transactions on Education*, vol. 55, no. 1, p. 99, 2012.
- [3] K. Schwaber and J. Sutherland, "The scrum guide," online, 2020. [Online]. Available: <https://scrumguides.org/scrum-guide.html>
- [4] V. Stray, D. I. Sjøberg, and T. Dybå, "The daily stand-up meeting: A grounded theory study," *Journal of Systems and Software*, vol. 114, pp. 101–124, 2016.
- [5] J. Pantichina, M. Mondini, D. Khanna, X. Wang, and P. Abrahamsson, "Are software startups applying agile practices? the state of the practice from a large survey," in *Agile Processes in Software Engineering and Extreme Programming: 18th International Conference, XP 2017, Cologne, Germany, May 22–26, 2017, Proceedings 18*. Springer International Publishing, 2017, pp. 167–183.
- [6] V. G. Stray, Y. Lindsjörn, and D. I. Sjøberg, "Obstacles to efficient daily meetings in agile development projects: A case study," in *2013 ACM / IEEE International Symposium on Empirical Software Engineering and Measurement*, 2013, pp. 95–102.
- [7] V. Stray, N. B. Moe, and D. I. Sjøberg, "Daily stand-up meetings: Start breaking the rules," *IEEE Software*, vol. 37, no. 3, pp. 70–77, 5 2020.
- [8] M.-L. Lähteenmäki and L. Uhlin, "Developing reflective practitioners through pbl in academic and practice environments," in *New approaches to problem-based learning*. Routledge, 2010, pp. 144–157.
- [9] J. A. Moon, *A handbook of reflective and experiential learning: Theory and practice*. Routledge, 2013.
- [10] D. A. Schön, *The reflective practitioner : how professionals think in action*. Basic Books, 1983.
- [11] F. Korthagen and A. Vasalos, "Levels in reflection: core reflection as a means to enhance professional growth," *Teachers and Teaching*, vol. 11, no. 1, p. 47–71, 2005.
- [12] K. Mann, J. Gordon, and A. MacLeod, "Reflection and reflective practice in health professions education: a systematic review," *Advances in Health Sciences Education*, vol. 14, no. 4, pp. 595–621, Nov. 2007.
- [13] D. F. Rico and H. H. Sayani, "Use of agile methods in software engineering education," in *2009 Agile conference*. IEEE, 2009, pp. 174–179.
- [14] L. Layman, L. Williams, K. Slaten, S. Berenson, and M. Vouk, "Addressing diverse needs through a balance of agile and plan-driven software development methodologies in the core software engineering course," *International Journal of Engineering Education*, vol. 24, no. 4, p. 659, 2008.
- [15] A. Shekar, "Active learning and reflection in product development engineering education," *European Journal of Engineering Education*, vol. 32, no. 2, pp. 125–133, 2007.
- [16] D. H. Schunk and B. J. Zimmerman, "Self-regulation and learning," *Handbook of Psychology, Second Edition*, vol. 7, 2012.
- [17] O. Hazzan, "The reflective practitioner perspective in software engineering education," *Journal of Systems and Software*, vol. 63, no. 3, pp. 161–171, 2002.
- [18] P. Salza, P. Musmarra, and F. Ferrucci, "Agile methodologies in education: A review," *Agile and lean concepts for teaching and learning: Bringing methodologies from industry to the classroom*, pp. 25–45, 01 2019.
- [19] O. Cico and L. Jaccheri, "Industry trends in software engineering education: A systematic mapping study," in *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, 2019, pp. 292–293.
- [20] S. Tenhunen, T. Männistö, M. Luukkainen, and P. Ihantola, "A systematic literature review of capstone courses in software engineering," *Information and Software Technology*, vol. 159, p. 107191, 2023.
- [21] M. Marques, S. F. Ochoa, M. C. Bastarrica, and F. J. Gutierrez, "Enhancing the student learning experience in software engineering project courses," *IEEE Transactions on Education*, vol. 61, no. 1, pp. 63–73, 2018.
- [22] R. Sandstø and C. Reme-Ness, "Agile practices and impacts on project success," *Journal of Engineering, Project & Production Management*, vol. 11, no. 3, 2021.
- [23] E. Derby, D. Larsen, and K. Schwaber, *Agile retrospectives: Making good teams great*. Pragmatic Bookshelf, 2006.
- [24] R. E. Landaeta, S. Viscardi, and A. Tolk, "Strategic management of scrum projects: An organizational learning perspective," in *First International Technology Management Conference*, 2011, pp. 651–656.
- [25] B. Cotrel, "Should your sprint retrospective be anonymous? why?" Aug 2022. [Online]. Available: <https://www.neatro.io/anonymous-retrospective/>
- [26] H. Burden and J.-P. Steghöfer, *Teaching and Fostering Reflection in Software Engineering Project Courses*. Singapore: Springer Singapore, 2019, pp. 231–262.
- [27] J. Babb, R. Hoda, and J. Norbjerg, "Embedding reflection and learning into agile software development," *IEEE Software*, vol. 31, no. 4, pp. 51–57, 2014.
- [28] M. Ryan, "The pedagogical balancing act: teaching reflection in higher education," *Teaching in Higher Education*, vol. 18, no. 2, pp. 144–155, 2013.
- [29] G. E. Mills and L. R. Gay, *Educational research: Competencies for analysis and applications*. ERIC, 2019.
- [30] W. Doyle and G. A. Ponder, "The practicality ethic in teacher decision-making," *Interchange*, vol. 8, no. 3, pp. 1–12, 1977.
- [31] R. Hake, "Interactive-engagement versus traditional methods: A six-thousand-student survey of mechanics test data for introductory physics courses," *Am. J. Phys.*, vol. 66, no. 1, pp. 64–74, January 1998.

# Automation of Test Skeletons within Test-Driven Development Projects

1<sup>st</sup> Muhammet Mustafa Olmez

*Dept. of Electrical and Computer Engineering  
North Carolina State University  
Raleigh, NC, USA  
molmez@ncsu.edu*

2<sup>nd</sup> Edward Gehringer

*Dept. of Computer Science  
North Carolina State University  
Raleigh, NC, USA  
efg@ncsu.edu*

**Abstract**—In addressing the need for test case generation in software projects and the validation and repair processes, various algorithms and AI models are increasingly being applied with novel approaches. On the other hand, despite the established effectiveness of the Test-Driven Development (TDD) approach in testing and development, there is still a lack of research examining the impact of human-machine interaction on software validation and coding.

This paper introduces a tool, the test-skeleton generator, which utilizes an OpenAI model to generate test skeletons. These skeletons include test names, signatures, and scenario descriptions, omitting the actual test bodies. To explore the implications of this tool, an empirical experiment involving student participation was conducted to assess the conversion of test skeletons into functional tests with human-machine interaction.

The study reveals significant insights, indicating that human-machine interaction plays a crucial role in shaping both the testing and programming phases, encouraging students to prioritize writing tests before modifying source code. Teams adopting this approach demonstrate a tendency to produce more tests, leading to higher code coverage. Additionally, our research underscores the growing potential of AI language models to generate tests that closely resemble those written by human developers. Notably, human-machine interaction has proven its significant positive impact on the validation and repair process of AI-generated tests.

**Index Terms**—Test skeletons, Test-driven development, test generation, Human-machine interaction, AI tool

## I. INTRODUCTION

Test-driven development (TDD) stands out as an agile software development approach utilized to enhance the overall quality of software products. The use of TDD has been extensively explored in various domains since 1950s [1]. Several studies have focused on evaluating the effectiveness of TDD in improving code quality [2], reducing defects [3], and enhancing design skills [4]. The challenges and benefits of implementing TDD have been discussed, considering factors such as participant experience and task complexity [5]. The integration of TDD is investigated with other development practices, such as continuous integration [6] and unit testing [7]. Through rigorous methodologies and empirical evaluations, studies aim to provide valuable insights into the practical implications and limitations of TDD, contributing to the advancement of software engineering practices.

While TDD has garnered significant attention and demonstrated several benefits, it is worth to highlight certain challenges and drawbacks associated with its implementation, especially in software education. [1] reported that the upfront preparation required for writing tests before code development can lead to a slower initial development pace. Additionally, inexperienced developers/participants may struggle to understand the TDD process and write effective tests, resulting in lower productivity and potential frustration. Moreover, adhering strictly to TDD may result in an excessive focus on test coverage and passing tests, potentially overshadowing other important aspects such as code design or system integration.

In contrast to students who begin small projects from scratch, students engaged in OSS-based course projects in a degree of computer science/engineering face the challenge of integrating their contributions into an existing codebase. Consequently, they must carefully consider how to ensure compatibility between their new code and the current design. However, the limited time allocated for these projects poses a significant obstacle for students to familiarize themselves adequately with their assigned project component. To address the time constraint challenge and facilitate adherence to the TDD approach, students can be provided with test skeletons. Test skeletons are basically test cases with test names, signatures and description of scenarios but not test bodies. These test skeletons served as a valuable resource, enabling students to gain a quicker and deeper understanding of the project requirements and objectives. By examining the given test skeletons, students were able to discern the desired outcomes and comprehended the project's scope more expeditiously. This aided them in effectively aligning their code contributions with the overarching project design, ultimately enhancing their ability to meet the project objectives within the allocated time frame [1]. The utilization of test skeletons as an instructional strategy proved instrumental in streamlining the students' comprehension and alignment with the TDD approach in the context of their assigned course projects.

Providing test skeletons to students working on open-source software projects as part of their degree program, particularly within the TDD approach, is beneficial [1]; however, creating the required test skeletons for writing tests before actual code development can be time-consuming, requiring a significant

amount of manual effort for instructors and mentors.

Over the past two decades, numerous methodologies have been proposed by researchers to automate software test generation, encompassing approaches utilizing UML diagrams [8], test names [9], search-based techniques [10], NLP-based methods [11], and AI-driven strategies [12]. Various tools such as EvoSuite [13], Randoop [14], and Pynguin [15] have been introduced to facilitate the generation of tests and test cases. Additionally, [16] has investigated multiple NLP-based algorithms, including tokenization, POS tagging, chunking and parsing, to automate the generation of test cases. It is evident that test cases generated automatically through these methods are more advantageous compared to manually created ones. Nevertheless, the validation process and the resolution of errors remain significant time-consuming challenges.

This paper focuses on the idea of generating test cases through human-machine interaction using the TDD approach. While there is no validation process in place during the creation of test skeletons, in this study, it is essential for humans to verify and assess them before and after integration into an Open Source Software (OSS) project. The setup is shown in Figure 1.

In this paper, we presented the test-skeleton generator, a tool designed to automate generating test skeletons for OSS projects within an Object-Oriented Development course. This tool, developed using Ruby, creates tests in *Rspec* format to align seamlessly with the technologies used in the projects under consideration. The generator tool utilizes *Regex* to parse the given data and sends prompts to the *OpenAI API* for generating test skeletons. With the ChatGPT 3.5 NLP algorithm [17], dedicated prompts, and the processed data, our results indicate that the test-skeleton generator reliably creates test scenarios with test names, signatures and descriptions resembling those in tests written by students in previous semesters. The tool generates tests skeletons as a guide for students, aiding them in writing tests and adopting TDD approach. Hence, the tool is designed to save time writing tests, understanding the project requirements, enabling instructors to concentrate more on the investigation of project outcomes.

This paper reports the findings of a quasi-experimental study conducted on the Expertiza Open-Source Software (OSS) project, an online peer-assessment tool utilized in courses offering student group projects [18]. In the course of Object Oriented Development, students were assigned projects involving existing files with minimal statement coverage or no tests, ensuring they had ample opportunities to write tests before refactoring or re-implementing the codebase. Auto-generated test skeletons were provided to help students in test writing and understanding the assigned project functions. The expectation was for students to complete auto-generated test skeletons, eliminate unnecessary scenarios, and incorporate additional relevant test cases in adherence to TDD principles. The study revealed no significant differences in the quality of source-code modifications between groups that utilized test skeletons and those that did not. However, the group using test skeletons added more tests and assertions, achieving higher

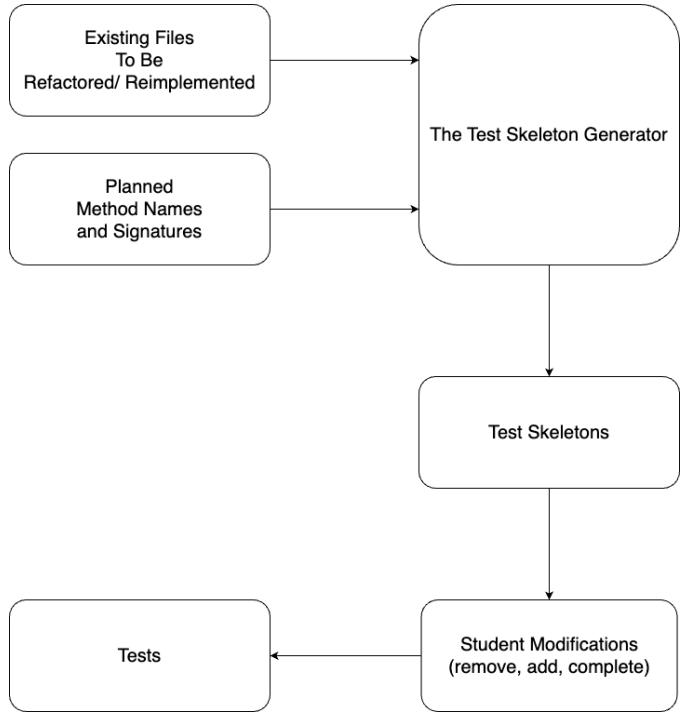


Fig. 1: Human-Machine Interaction in operation

statement coverage compared to the groups that did not use test skeletons, on average. Furthermore, the incorporation of auto-generated test skeletons resulted in time savings, enabling students to complete their last commits ahead of schedule. Statistical analysis has been conducted, and the results align with the experimental findings in this study.

## II. BACKGROUND

### A. Test-Driven Development Approach

Test-driven development (TDD) has a rich history that can be traced back to the 1950s [19]. Since then, TDD has evolved into a formal iterative process where developers start by writing failing tests for the code they are about to develop. They then proceed to implement the necessary functionality to make the tests pass. Alternatively, if the initial tests pass, developers can directly focus on refactoring existing code, ensuring that the tests continue to pass throughout the development process.

Recognizing the potential advantages of TDD, educators have integrated it into numerous computer science courses, aiming to equip students with the ability to produce code of superior quality with reduced defects [1]. To assess the efficacy of the TDD approach in entry-level computer science courses, several researchers conducted structured experiments and evaluated its impact. Some experimental findings indicated that the TDD approach indeed facilitated the creation of high-quality code and enhanced design skills [1], [20], [21] while contrasting conclusions were drawn by other researcher who asserted that the TDD approach did not consistently yield high-quality code [22]. Alternatively, integrating the TDD process with other methodologies, such as mutation testing, empowers

developers to create more robust tests and produce higher-quality code [23].

#### B. Test Skeletons

Test skeletons play a fundamental role in software testing, providing a structured framework for creating effective test suites. They serve as a guideline or template, outlining the structure and format of the tests that need to be implemented. Test skeletons are essentially test methods that include test names, signatures, and scenario descriptions, but do not contain test bodies. [1].

By having predefined test skeletons, developers/students can focus on implementing the test logic and writing the necessary code to fulfill the requirements, rather than spending time figuring out the test structure. By following established patterns and conventions within the test skeletons, developers can ensure a standardized approach to testing across the project or team. Test skeletons facilitate the adoption of a TDD methodology, where tests are first auto-generated and then written by humans before the implementation code. This approach helps developers/students define the desired behavior and requirements upfront and assists in identifying any discrepancies or issues early in the development process. Moreover, test skeletons partially assist developers in grasping the functionality of the task or project without delving into the source code.

#### C. OSS Projects

In the Object-Oriented Development course, students are assigned projects that involve active participation with Expertiza, an OSS platform [18]. The evaluation of their contributions is based on the examination of pull requests on GitHub and a wiki-page created by the students to document their project modifications and implementations. The primary objectives of these projects are to comprehend the source code, refactor specific project components, or re-implement existing functionality. Students are encouraged to form teams of 2 or 3 members for each project, with the flexibility to switch between teams [1].

This study focuses on participants who are actively engaged in OSS projects, and the OSS project outcomes, the code refactored or re-implemented, gathered from the team's GitHub repository and project wiki-pages.

#### D. OpenAI API

The Open AI API, a product of Open AI, facilitates the integration of advanced natural language processing models into applications, offering functionalities like text completion and sentiment analysis [24]. Users require an API key, and costs may be associated based on usage. Despite a free version with limitations, various pricing plans accommodate different needs.

#### E. Human-Machine Interaction

1) *Interaction with AI:* Effectively interacting with AI requires adherence to specific design guidelines encompassing

stages such as clearly conveying intentions, understanding system capabilities, following instructions, providing relevant context and prompts, and adapting over time. These guidelines span from the initial phase to during interaction, when errors occur, and over time, ensuring a seamless and meaningful communication process [25]. Recognizing AI system capabilities and limitations is crucial for setting realistic expectations and customizing interactions accordingly. Adhering to instructions and prompts ensures precision, while offering correct context enhances the AI system's understanding of user needs. Following these principles enables users to engage in appropriate and efficient interactions with AI systems.

Precision in formulating questions and providing relevant context is crucial for obtaining accurate and relevant responses from AI [27]. Clear intentions and well-structured queries facilitate efficient communication, leading to faster and more precise answers. Additionally, asking unbiased questions promotes fairness in AI system outputs, maximizing their utility for users.

2) *Human Factor on Verification:* While AI systems have advanced capabilities, there are certain areas where human expertise continues to play a significant role. Humans can complement AI in various ways, especially in testing.

In the training phase, large language models such as *OpenAI API* frequently focus on maximizing likelihood objectives, which can lead to the production of text that lacks coherence or consistency—an occurrence termed as hallucination. This highlights the pivotal role of fact-verification as an essential feature for effective detection [26]. Several methods are proposed to handle this lack of coherence and errors due to inconsistency of trained large language models. [27] is asking multiple questions in one chat until the context is destroyed so the last answer is the most consistent one. Other methods such as retrieval augmentation [28] and flip-flop language modelling [29] have been presented. However, these methods still leave errors behind. In this scenario, the optimal approach for managing errors and hallucinations with tools powered by AI language models requires human interaction during the verification and error-handling stages.

Considering that test skeletons are automatically created with an AI-based tool, the test-skeleton generator, it is inevitable that there might be incorrect test scenarios or test descriptions. This practice encourages students to take a TDD approach, as test skeletons are provided to students before they start software development, and for verification and correction, students are asked to correct these errors and make the tests work.

### III. EXPERIMENTAL SETUP

#### A. Dataset

The subjects of our experimental study are students enrolled in the Object Oriented Development course during the ongoing semester within the Computer Science/Engineering program.

An email is sent and an announcement is made to the entire class, encouraging students to take part in our study and embrace the TDD approach for their projects. Out of

the 84 students, 43 opt in. Further investigation reveals that 12 students assigned to front-end projects are unresponsive. This lack of response might suggest their awareness that test skeletons are unnecessary for such projects, especially since using test skeletons is neither mandatory nor graded in the course.

In terms of team participation, 17 teams participated in our study and 8 of them used the test skeletons provided. It is worth mentioning that, however, on 3 teams, some students expressed interest in participating, but not all team members agreed. Therefore, we had to exclude these teams from the study.

Throughout the study, data were gathered assessing the utilization of auto-generated test skeletons for OSS projects by analyzing student GitHub repositories. Evaluations were conducted by the researchers manually observing GitHub repositories and engaging in communication with students. Researchers manually recorded metrics including the number of tests, the number of lines in test code, the number of assertions, and the total number of tests and assertions added by students. By adopting this approach, unexpected occurrences like whole-file changes, last-minute submissions, and instances of copying and pasting code can be mitigated. Furthermore, the manual recording process affords researchers the opportunity to scrutinize the code's quality and assess adherence to established standards in coding practices. Additionally, two surveys [30] were conducted anonymously: a pre-survey to understand students' backgrounds and experiences, and a post-survey deployed to obtain feedback on test skeletons, TDD and our research. At the end of the study, a focus group was organized to delve into the survey responses. Individual Zoom meetings were conducted with participants to discuss survey answers and gather their insights.

The test-skeleton generator utilized either the latest files available in Expertiza or relied on the methods and signatures provided by students to create test skeletons. In order to evaluate the effectiveness, correctness and robustness of the test-skeleton generator tool, researchers went through a comprehensive history of Expertiza OSS Projects dating back to 2007. Similarities of tests and test scenarios between auto-generated test skeletons and human-written tests were identified.

### B. The Test-Skeleton Generator Tool

Various tools have been developed to address test generation in the testing area, utilizing diverse algorithms [11], [12], [31], approaches [32], [33], and extensions [34], [35]. Some tools are designed for specific tasks, but they may need updates or modifications to accommodate novel methods or specific objectives in time. Recently, tools utilizing the OpenAI models have shown promising results, although they require thorough validation, correction and repair processes [34], [36]. Assessing the strengths and weaknesses of existing tools, there is a need for a new tool designed to collaborate with humans in the test-case generation process. Hence, integrating automated test case generation with human involvement for tasks such

```

describe "#finished_quiz" do
context "when a participant has finished a quiz" do
  it "retrieves the participant's response for the quiz" do
    # Test code
  end

  it "retrieves the quiz response map for the given map_id" do
    # Test code
  end

  it "retrieves the quiz questions associated with the quiz response map" do
    # Test code
  end

  it "retrieves the participant who attempted the quiz" do
    # Test code
  end

  it "retrieves the quiz score for the participant" do
    # Test code
  end
end

```

(a) Auto-generated test skeletons

```

describe "#finished_quiz" do
context "when a participant has finished a quiz" do
  it "retrieves the participant's response for the quiz" do
    # Test code
    expect(Response).to receive_message_chain(:where,:first)
    controller.finished_quiz
  end

  it "retrieves the quiz response map for the given map_id" do
    # Test code
    expect(QuizResponseMap).to receive(:find)
    controller.finished_quiz
  end

  it "retrieves the quiz questions associated with the quiz response map" do
    # Test code
    expect(Question).to receive(:where)
    controller.finished_quiz
  end

  it "retrieves the participant who attempted the quiz" do
    # Test code
    expect(AssignmentTeam).to receive(:find)
    controller.finished_quiz
  end

  it "retrieves the quiz score for the participant" do
    # Test code
    expect(quiz_response_map.quiz_score).to eq(quiz_response_map.quiz_score)
    controller.finished_quiz
  end
end

```

(b) Tests completed by students

Fig. 2: Auto-generated test skeletons (2a) and tests filled by students(2b)

as filling, correction, and verification is essential and yield valuable outcomes.

The test-skeleton generator tool is designed to automate the generation of test skeletons within the context of *Rspec* testing in Ruby on Rails. Its primary objective is to autonomously generate test skeletons, leaving the implementation of test bodies to students. The generator operates through three sequential steps, involving the reading of a Ruby file, communicating with OpenAI API and the subsequent generation of test skeletons in a corresponding spec file.

In the initial step, the tool reads a Ruby file (models, controllers, or helpers) and utilizes Regular Expressions to

extract information such as class names, method names, and method signatures.

On the second step, upon identifying each method, the generator communicates with the OpenAI API. It posts tailored questions related to the extracted methods, seeking more reliable responses from the API. The received responses consist of test skeletons generated for the specified methods.

In the third step, with the auto-generated test skeletons, the tool constructs a spec file. The spec file adopts the name of the original Ruby file and incorporates relevant class information, method names, method signatures, and the associated test skeletons.

The Ruby files intended for reading are associated with student projects. These files should be modified (refactored/re-implemented) and tested based on the project specifications. In cases where related files are absent or not found, we request the inclusion of potential or planned method names and signatures in a Ruby file at the project's outset. These are extracted from functions or project descriptions by students. This allows the tool to read the file and generate test skeletons just with method names and signatures.

The test-skeleton generator tool utilizes “gpt-3.5-turbo-16k”. It is a model trained until September 2021 [24]. In the model, there’s a hyperparameter known as “temperature” that plays a role in determining token probabilities when generating output with the large language model. This temperature parameter has a range of 0 to 2, with lower values promoting greater predictability and higher values introducing more variability into the generated content. The parameter is set to 0 to get same results for identical methods each time. Additionally, we added a 20-second delay between each post, which serves as a question for each method. This adjustment was made to align with the limitations of the free version of GPT model, which permits only three questions per minute. The delay can be set to zero if the user opts for a premium account, incurring associated costs depending on the tokens used [34]. However, it’s important to note that in this study, we did not factor in the cost considerations, as the free version proved to be efficient in generating test skeletons.

The effectiveness of AI model communication relies significantly on the choice of prompts and questions, as responses can vary based on the provided prompts [34]. We experimented with several combinations of prompts and examined fundamental language usage, addressed error handling, adapted the model, and prioritized achieving results and goals. The most favorable outcomes were obtained using the following prompt:

**Prompt.** Create test scenarios with method names and descriptions for the given Ruby method, organized in a spec file format,

Do not include any extension of test bodies,  
Do not include RSpec line at the beginning.

Once the spec file is generated with test skeletons, the file can be committed directly to the spec folder in students’ GitHub repositories. There is no necessity to modify, repair,

```
(a)
1 describe "select_mentor" do
2   context "when there are mentors available for the
     assignment" do
3     it "returns the mentor with the lowest team count
       for the given assignment" do
4       # Test scenario 1
5       # Given an assignment_id
6       # When there are multiple mentors with different
         team counts for the assignment
7       # Then it should return the mentor with the
         lowest team count
8
9
10      # Test scenario 2
11      # Given an assignment_id
12      # When there are multiple mentors with the same
        lowest team count for the assignment
13      # Then it should return the first mentor in the
        list
14
15      # Test scenario 3
16      # Given an assignment_id
17      # When there is only one mentor available for
        the assignment
18      # Then it should return that mentor
19    end
20  end

(b)
1 describe '#select_mentor' do
2   it 'returns the mentor with the fewest teams they
     mentor' do
3     allow(MentorManagement).to receive(:zip_mentors_with_team_count)
4       .with(assignment.id)
5       .and_return([mentor.id, 0])
6     allow(User).to receive(:where).with(id: mentor.id).
      and_return([mentor])
7     mentor_user = MentorManagement.select_mentor
      assignment.id
8     expect(mentor_user).to eq mentor
9   end
10 end
```

Fig. 3: Similarity between auto-generated test skeletons (a) and student-written test cases in past (b)

or validate the auto-generated test skeletons, as these tasks are expected to be undertaken by students. Figure 2 illustrates the modifications made by students. If a file with the same name already exists, we send the test skeletons via a spreadsheet. This allows students to compare methods, ignore duplicates, and pull in the remaining methods as needed. This step supports early adherence to the Test-Driven Development approach in their projects.

The test quality of the tool is evaluated by comparing its performance with test scenarios from previous semesters written by students. Test skeletons are created for 40 different spec files across various projects. The similarity between the number of automatically generated test scenarios and pre-existing student-written test scenarios is determined to be 60.8%.

As shown in Figure 3, both the auto-generated and student-written tests share identical method names and signatures, directly derived from the source file. The method and scenario descriptions share comparable language and convey equivalent

meanings in both the auto-generated and student-written tests, reflecting their direct adaptation from the source file. Notably, the auto-generated version provides additional details, enhancing its utility by offering more comprehensive explanations to aid students in understanding the source code's functionality.

The test-skeleton generator has proven to be a valuable tool in automating the creation of dependable and consistent test skeletons, resulting in time savings across more than 40 project files. It produces test cases that closely resemble those generated by humans. Equipped with basic language, instructions, and supplementary descriptions, it serves as a helpful resource for students in comprehending the functionality of source code before commencing their projects. Moreover, it has facilitated students in adhering to the Test-Driven Development (TDD) approach.

#### IV. EVALUATION

##### A. Experimental Results

In this section, we address the following research questions:

- RQ1.** Does the test-skeleton generator generate test stubs that can be turned into real tests?
- RQ2.** Do the auto-generated test skeletons help students comply with TDD?
- RQ3.** Does the TDD approach help students write more tests and assertions with an higher code coverage?
- RQ4.** How does the adoption of TDD impact the dynamics of human-machine interaction in software testing/development processes?

In assessing the effectiveness of the test-skeleton generator within the Test-Driven Development (TDD) approach, we take into account the total count of test scenarios, the comparison of similar test scenarios between auto-generated and student-written code, and a manual assessment of the overall quality of the tests.

Figure 4 shows that, on average, the test-skeleton generator produced 29% more test scenarios for 40 different project files compared to student-written test cases in prior semesters. Furthermore, the auto-generated test skeletons demonstrated a higher count of test scenarios when contrasted with student-written test cases in the same 40 project files. On average, the auto-generated test skeletons had a 46.75% similarity to the test cases composed by students.

We conducted a manual examination of pull requests submitted by teams to gather study metrics. The number of tests and assertions contributed by students was documented through cross-verification by researchers. Two researchers split the pull requests of projects into two parts and made an online spreadsheet to note down study details. Each researcher filled out their section first without checking the other's entries. Then, they switched parts and completed the remaining section independently. After collecting all the data, they discussed any differences in their findings. They looked at the pull requests to figure out mistakes in the values they recorded. The data was collected and reviewed by both researchers. Figure 5 illustrates that teams utilizing test skeletons (5a) generated a

greater number of tests and assertions compared to those that did not use test skeletons (5b). On average, teams utilizing test skeletons added 32 tests and 45 assertions, while teams that did not use test skeletons added 8 tests and 10 assertions. Furthermore, two teams refrained from adding any new tests or assertions; instead, they fulfilled their testing requirements by refactoring existing test files. Researchers did not observe any substantial disparity in the quality of tests.

Code coverage is a crucial metric for assessing the outcomes of OSS projects, and it has been established that Test-Driven Development (TDD) contributes to higher code coverage among students [1]. In this study, students made modifications to source code which has code coverage rates of 71.12% for refactoring projects and 63.71% for reimplementation projects. Figure 7 indicates that teams utilizing test skeletons achieved higher code coverage, ranging from a minimum of 74.07% to a maximum of 80.22%. Conversely, teams not utilizing test skeletons demonstrated lower code coverage, with rates ranging from a minimum of 69.76% to a maximum of 74.93%.

In comparing two independent groups, Welch's *t*-test serves as a statistical analysis method when the variances of the groups are unequal. In this study, Welch's *t*-test was employed to assess the significance between teams that used test skeletons and teams that did not. A Python script was designed using the *scipy* library, importing the *ttest\_ind* which is the Welch's *t*-test algorithm. The statistical analysis focused on the data regarding the number of tests and assertions added by students.

The results, illustrated in Figure 6, reveal that for the number of tests added, the *t*-statistic equals 3.4476 with a corresponding *p*-value of 0.0078. Similarly, for the number of assertions added, the *t*-statistic is 4.2955 with a *p*-value of 0.0017. Error bars, representing standard deviation and reflecting the variability within each group, are depicted. The obtained *p*-values are below the commonly accepted significance level of 0.05.

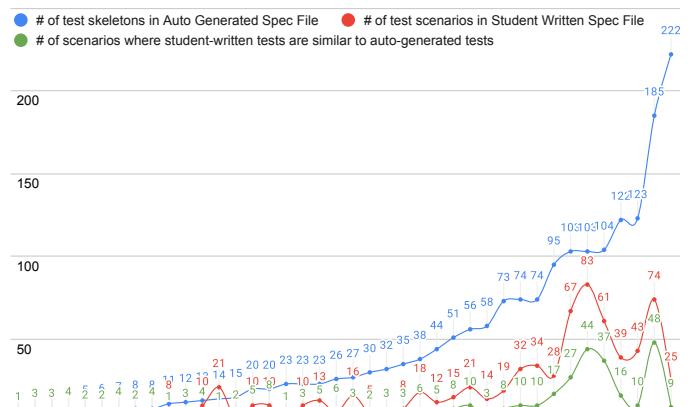
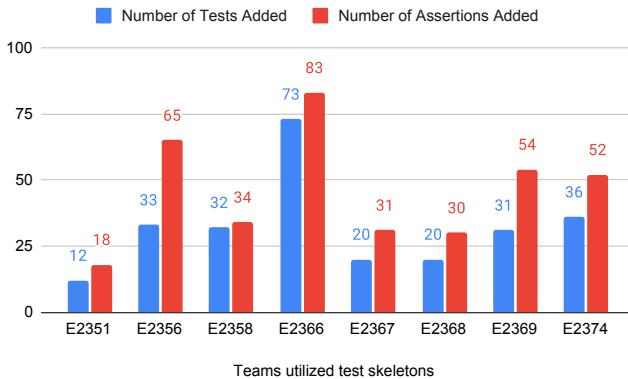
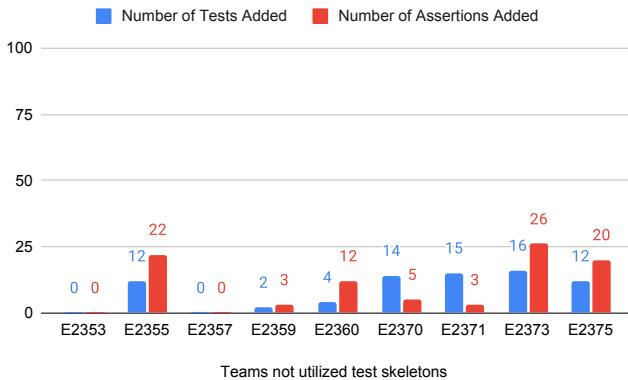


Fig. 4: Number of Similar Test Scenarios between auto-generated test skeletons and student-written test scenarios in past for 40 different project files (sorted by # of test skeletons in auto-generated spec file)



(a) Teams that utilized test skeletons



(b) Teams that did not utilize test skeletons

Fig. 5: Number of tests and assertions added by teams that utilized test skeletons (5a) versus those that did not utilize test skeletons (5b)

Consequently, we have enough evidence to assert a statistically significant difference in the mean effectiveness between teams used test skeletons and teams that did not. The positive t-statistic suggests that, on average, teams used test skeletons exhibit a higher level of effectiveness compared to those that did not. The statistical analysis further validates our

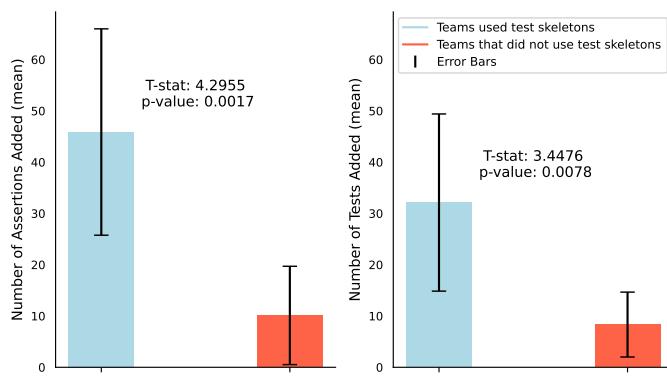
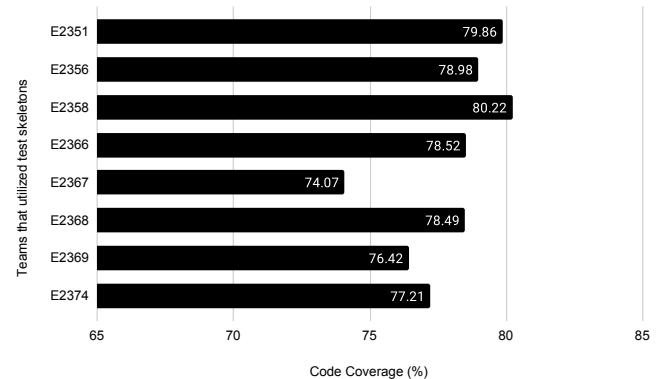
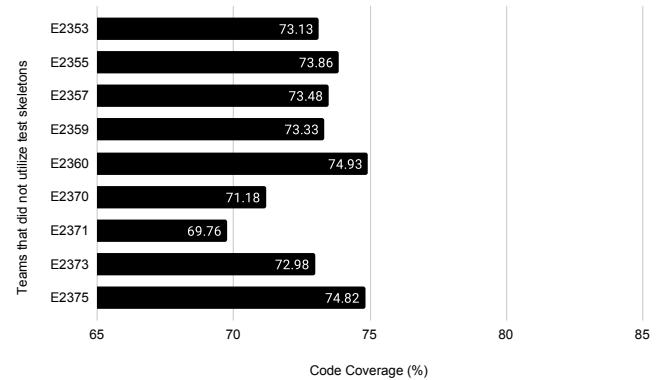


Fig. 6: Welch's t-test results



(a) Code Coverage for teams that utilized test skeletons



(b) Code Coverage for teams that did not utilize test skeletons

Fig. 7: Code Coverage achieved by teams that utilized test skeletons (7a) versus those that did not utilize test skeletons (7b)

experimental findings.

Based on our analysis, the use of auto-generated test skeletons contribute to boosting students' test-writing activities, resulting in a higher number of tests, assertions, and overall code coverage. Students, when using auto-generated test skeletons, feel more motivated to generate tests, and this motivation extends beyond only code refactoring. Significantly, observations from the initial and final commits on GitHub demonstrate that students using auto-generated test skeletons are more likely to adhere to the Test-Driven Development (TDD) process.

### B. Survey Results

The pre-survey garnered a response rate exceeding 32%, with 27 out of 84 students responding. Analysis revealed that over 25% of the students had more than one year of industry work experience, while more than 48% had prior experience in writing automated tests. Additionally, 40% of students felt they had a knowledgeable level of expertise. The post-survey, with an almost 37.5% response rate (9 out of 24), was conducted among teams utilizing test skeletons. Over half of the students found test skeletons useful for test writing and TDD compliance, and 44% believed they improved test quality. Feedback

indicated that while test skeletons covered a broad range of topics, some students found certain scenarios unrelated and time-consuming to remove. Existing tests posed challenges in identifying and eliminating duplicates, prompting students to actively work on test cases to address these issues. The aim was to encourage students to focus on tests before development within the TDD framework.

### C. Focus Group Results

We established a focus group to delve into discussions surrounding auto-generated test skeletons, the Test-Driven Development (TDD) framework, and responses to surveys. The focus group comprises four individuals, with one participant adhering to TDD without using test skeletons, and another utilizing test skeletons in conjunction with TDD.

Individual meetings were scheduled with each focus group member to document discussions and gather their insights. In the case of the participant who did not use test skeletons, he expressed the view that such skeletons would be beneficial. The reason his team abstained from using test skeletons was an authentication issue encountered while testing controllers. Despite this hurdle, they opted to create their own tests before initiating the application, subsequently running these tests through the Postman API.

Participants were asked about problems they faced while running test skeletons. Everyone agreed that issues like authentication for controllers and irrelevant test scenarios were a challenge. Further discussions revealed that these unrelated scenarios actually helped participants in adhering to Test-Driven Development (TDD). Participants found that these scenarios forced them to figure out which tests were relevant by looking at the source code, leading to a better understanding of which classes and variables needed to be re-implemented or refactored.

Students may be unfamiliar with their projects requirements and environments until their first meetings with mentors. When began to working on tests first, participants found it necessary to go back and look at the source code to understand project requirements. This study highlights that having auto-generated test skeletons significantly helped students understand the source code better. Additionally, based on all participant responses, it's clear that this approach saved them time in completing tasks.

### D. Overall Results

Regarding **RQ1**, the findings showed that the test-skeleton generator produces skeletons that are similar to test scenarios compared to student-written test cases in previous semesters. Teams utilized relevant test skeletons for their projects and successfully converted them into passed tests. Additionally, a participant in the focus group who declined to use test skeletons mentioned finding test skeletons encouraging for writing tests first. Students commented that the test skeletons were beneficial for their projects.

Regarding **RQ2**, monitoring the teams' pull requests and repositories revealed that students added auto-generated test

skeletons to their repositories at the project's outset. Providing testing guidelines—referred to as test skeletons in advance—encouraged them to revisit the source code, leading to subsequent modifications of tests for successful execution. Our conclusion, supported by surveys and focus group discussions, is that this process aided students in understanding project functions and complying with Test-Driven Development.

Regarding **RQ3**, teams that adopted the TDD framework achieved higher code coverage by adding more tests and assertions during the process of reimplementation or refactoring. While teams that did not adhere to TDD also increased code coverage by adding tests and assertions, the improvement was not as significant as observed in teams adopting TDD. Therefore, our findings demonstrate that adopting TDD resulted in improved code coverage in their projects.

Regarding **RQ4**, our approach involves engaging students with auto-generated test skeletons to ensure reliable, repaired and comprehensive tests. To achieve this, we run the test-skeleton generator on our end at the beginning of the OSS projects and provide auto-generated test skeletons to the students, requiring them to convert them into passed test cases. Findings from our experiment, surveys, and focus group discussions demonstrate that students removed unrelated test scenarios, added extra scenarios as needed, avoided duplicated methods, and filled skeletons. Consequently, they needed to revisit the source code to be tested and modify it when necessary. As a result, they managed to create auto-generated but human-verified tests with higher coverage. Hence, we can conclude that human-machine interaction, when adopting TDD, positively impacted testing and development.

### E. Threats to Validity

The participation rate was somewhat less than 50% of the class, and it is not known how representative it was of the class as a whole. The emphasis placed on testing in the class, and the requirements for thoroughly testing projects may have induced students to write more tests than they would in another setting. Furthermore, those who did participate in the project may have skewed the sample toward students who are more interested in the testing process. We considered using the previous semester as a control group, since use of test skeletons was minimal in that semester. However, a low opt-in rate left us with only six teams, too small to be an effective control group.

## V. DISCUSSION

*Challenges:* During the experiment, a hurdle arose when students encountered difficulties running tests for controllers due to authentication requirements. This obstacle hindered their ability to prioritize test-driven development initially. However, students successfully overcame this hurdle by utilizing the Postman API to execute the tests.

An additional challenge faced was recruiting a focus group. Unlike anonymous and short surveys, the focus group necessitated in-depth discussions and sought valuable insights from participants. Given that a majority of our participants were master's degree students juggling numerous assignments

throughout the semester, it was challenging to motivate them to willingly engage in these discussions within their demanding schedules. To overcome this obstacle, we effectively addressed it by sending individual invitation emails to students who were actively participated in the study and asked additional questions to pique their interest.

*Limitations.*: The test-skeleton generator tool is configured to utilize the free version of the OpenAI API, resulting in a 20-second delay between each query. This delay is imposed by the free version's rate limitation, allowing users only three queries per minute. Transitioning to the unlimited version of the API would incur costs. Moreover, considering the dynamic nature of advancements in technology, the research acknowledges the release of novel models from the same company during our study. It's important to note that updates to the test-skeleton generator may be necessary to align with any new releases announced by OpenAI.

## VI. RELATED WORK

Test-driven development (TDD) has emerged as a pivotal agile software development approach with a history dating back to the 1950s [19]. Numerous studies have delved into its effectiveness, examining aspects such as code quality improvement [2], defect reduction [3], and enhanced design skills [4]. TDD's integration with other practices like continuous integration [6] and unit testing [7] has also been explored to provide comprehensive insights into its practical implications. Integrating the TDD process with methodologies such as mutation testing has been explored to empower developers to create more robust tests and produce higher-quality code [23].

Recognizing TDD's potential advantages, educators have integrated it into computer science courses to enhance code quality and reduce defects [1]. Structured experiments assessing the efficacy of TDD in entry-level computer science courses have yielded varying conclusions [1], [20]–[22]. Some studies reported positive impacts on code quality and design skills, while others asserted inconsistent results. Regarding code completion, on the other hand, software projects adopting TDD can be used to automate tests [37].

While TDD has demonstrated various benefits, challenges persist, especially in software education. The upfront preparation required for writing tests before code development can initially slow down the development pace. However, providing test skeletons within the TDD approach is reported to be beneficial for students [1].

In response to the challenges associated with manual test creation, researchers have proposed methodologies to automate software test generation such as mutation testing, UML diagrams, AI-driven methods and models [8]–[12]. Tools like EvoSuite, Randoop, and Pynguin have been introduced to facilitate test and test case generation [13]–[15]. Although automated methods offer advantages, challenges persist in the validation process, error resolution and bug-finding ability [34], [36].

Effective interaction with AI requires adherence to design guidelines, including conveying intentions, comprehending

system capabilities, adhering to instructions, and adapting over time. Understanding the various phases of interaction with AI, including the "initial" phase, "during interaction," "when wrong," and "over time," is crucial for effective communication [25].

Large language models like OpenAI API focus on maximizing likelihood objectives during the training phase, potentially leading to text hallucination. Fact-verification emerges as a crucial feature for effective detection of inconsistencies and ensuring coherence in generated text [33]. Schafer et al. [31] indicate that utilizing Language Models (LLMs) for test case automation yields positive outcomes. Nonetheless, the approach remains deficient in effectively managing errors and identifying useful failing tests.

Jalil et al. [27] suggest a strategy involving the submission of multiple questions in a sequential manner within a single chat session. This method continues until the chat is terminated, ensuring that the final response is the most consistent one. Another proposed method involves retrieval augmentation, as outlined by Jalil. This approach seeks to improve coherence by enhancing the model's ability to retrieve relevant information. Liu et al. [29] introduce the concept of flip-flop language modeling as an additional method. This approach aims to address inconsistencies by modifying the language model's dynamics during the generation process.

## VII. CONCLUSION AND FUTURE WORK

This research introduced a test-skeleton generator that created scenarios resembling previously written student test cases. Teams effectively utilized these skeletons by transforming them into successful tests. Offering testing guidelines beforehand encouraged students to incorporate auto-generated test skeletons, facilitating a revisiting of source code for successful TDD. The research teams' monitoring of pull requests and repositories revealed the early integration of skeletons, with teams adopting TDD achieving higher code coverage through the addition of tests and assertions during reimplementation or refactoring. The study's methodology involved students actively engaging with auto-generated skeletons, resulting in human-verified tests with enhanced coverage, number of tests and assertions highlighting the positive impact of human-machine interaction in TDD on testing and development. Participants, including those who did not use skeletons, found them helpful in initiating the test-writing process.

We have determined that utilizing human-machine interaction in testing serves as a robust method to validate AI-generated test skeletons, while also functioning as a connecting tool that bridges the gap between students and the Test-Driven Development framework.

Examining the cost of multiple Large Language Models (LLMs) and investigating the impact of prompts could improve the efficiency and effectiveness of human-machine interactions in testing. Future work could also integrate structural coverages like Branch and Modified Condition/Decision (MC/DC) coverage and incorporate black-box techniques such as Boundary Value and Input Domain coverage. Leveraging Design by

Contract [38] principles for test-case derivation would further strengthen the testing framework. These aspects are left for future work.

#### DATA AVAILABILITY STATEMENT

The open-source implementation of the test-skeleton generator is available on GitHub.<sup>1</sup> Experimental raw data is available in Zenodo<sup>2</sup>

#### REFERENCES

- [1] Z. Hu, Y. Song, and E. F. Gehringer, “A test-driven approach to improving student contributions to open-source projects,” in *2019 IEEE Frontiers in Education Conference (FIE)*, pp. 1–9, IEEE, 2019.
- [2] T. Dogša and D. Batič, “The effectiveness of test-driven development: an industrial case study,” *Software Quality Journal*, vol. 19, pp. 643–661, 2011.
- [3] J. W. Wilkerson, J. F. Nunamaker, and R. Mercer, “Comparing the defect reduction benefits of code inspection and test-driven development,” *IEEE transactions on software engineering*, vol. 38, no. 3, pp. 547–560, 2011.
- [4] A. Gupta and P. Jalote, “An experimental evaluation of the effectiveness and efficiency of the test driven development,” in *First International Symposium on Empirical Software Engineering and Measurement (ESEM 2007)*, pp. 285–294, IEEE, 2007.
- [5] A. Santos, S. Vegas, O. Dieste, F. Uyaguari, A. Tosun, D. Fucci, B. Turhan, G. Scanniello, S. Romano, I. Karac, et al., “A family of experiments on test-driven development,” *Empirical Software Engineering*, vol. 26, pp. 1–53, 2021.
- [6] C. Matthies, A. Treffer, and M. Uflacker, “Prof. ci: Employing continuous integration services and github workflows to teach test-driven development,” in *2017 IEEE Frontiers in Education Conference (FIE)*, pp. 1–8, IEEE, 2017.
- [7] A. Tosun, M. Ahmed, B. Turhan, and N. Juristo, “On the effectiveness of unit tests in test-driven development,” in *Proceedings of the 2018 International Conference on Software and System Process*, pp. 113–122, 2018.
- [8] A. Cavarra, C. Crichton, J. Davies, A. Hartman, and L. Mounier, “Using uml for automatic test generation,” in *Proceedings of ISSTA*, vol. 15, 2002.
- [9] B. Zhang, E. Hill, and J. Clause, “Automatically generating test templates from test names (n),” in *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 506–511, IEEE, 2015.
- [10] P. McMinn, “Search-based software test data generation: a survey,” *Software testing, Verification and reliability*, vol. 14, no. 2, pp. 105–156, 2004.
- [11] C. Wang, F. Pastore, A. Goknil, and L. C. Briand, “Automatic generation of acceptance test cases from use case specifications: an nlp-based approach,” *IEEE Transactions on Software Engineering*, vol. 48, no. 2, pp. 585–616, 2020.
- [12] D. Santiago, T. M. King, and P. Clarke, “Ai-driven test generation: machines learning from human testers,” in *Proceedings of the 36th Pacific NW Software Quality Conference*, pp. 1–14, 2018.
- [13] G. Fraser and A. Arcuri, “Evosuite: automatic test suite generation for object-oriented software,” in *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*, pp. 416–419, 2011.
- [14] C. Pacheco, S. K. Lahiri, M. D. Ernst, and T. Ball, “Feedback-directed random test generation,” in *29th International Conference on Software Engineering (ICSE’07)*, pp. 75–84, IEEE, 2007.
- [15] S. Lukasczyk and G. Fraser, “Pynguin: Automated unit test generation for python,” in *Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: Companion Proceedings*, pp. 168–172, 2022.
- [16] I. Ahsan, W. H. Butt, M. A. Ahmed, and M. W. Anwar, “A comprehensive investigation of natural language processing techniques and tools to generate automated test cases,” in *Proceedings of the Second International Conference on Internet of things, Data and Cloud Computing*, pp. 1–10, 2017.
- [17] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al., “Language models are few-shot learners,” *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.
- [18] E. Gehringer, L. Ehresman, S. G. Conger, and P. Wagle, “Reusable learning objects through peer review: The expertiza approach,” *Innovate: Journal of Online Education*, vol. 3, no. 5, p. 4, 2007.
- [19] D. D. McCracken, *Digital computer programming*. John Wiley & Sons, Inc., 1957.
- [20] V. S. Bhaduria, R. K. Mahapatra, and S. P. Nerur, “Performance outcomes of test-driven development: an experimental investigation,” *Journal of the Association for Information Systems*, vol. 21, no. 4, p. 2, 2020.
- [21] M. T. Baldassarre, D. Caivano, D. Fucci, N. Juristo, S. Romano, G. Scanniello, and B. Turhan, “Studying test-driven development and its retention over a six-month time span,” *Journal of Systems and Software*, vol. 176, p. 110937, 2021.
- [22] D. Fucci, H. Erdogmus, B. Turhan, M. Oivo, and N. Juristo, “A dissection of the test-driven development process: Does it really matter to test-first or to test-last?,” *IEEE Transactions on Software Engineering*, vol. 43, no. 7, pp. 597–614, 2016.
- [23] A. Roman and M. Mnich, “Test-driven development with mutation testing—an experimental study,” *Software Quality Journal*, vol. 29, pp. 1–38, 2021.
- [24] OpenAI, “Openai gpt-3 api.” <https://platform.openai.com/docs/models/gpt-3-5>, 2022. Version gpt-3.5-turbo-16k, Accessed: December, 2023.
- [25] S. Amershi, D. Weld, M. Vorvoreanu, A.ourney, B. Nushi, P. Collisson, J. Suh, S. Iqbal, P. N. Bennett, K. Inkpen, et al., “Guidelines for human-ai interaction,” in *Proceedings of the 2019 chi conference on human factors in computing systems*, pp. 1–13, 2019.
- [26] R. Tang, Y.-N. Chuang, and X. Hu, “The science of detecting llm-generated texts,” *arXiv preprint arXiv:2303.07205*, 2023.
- [27] S. Jalil, S. Rafi, T. D. LaToza, K. Moran, and W. Lam, “Chatgpt and software testing education: Promises & perils,” in *2023 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pp. 4130–4137, IEEE, 2023.
- [28] K. Shuster, S. Poff, M. Chen, D. Kiela, and J. Weston, “Retrieval augmentation reduces hallucination in conversation,” *arXiv preprint arXiv:2104.07567*, 2021.
- [29] B. Liu, J. T. Ash, S. Goel, A. Krishnamurthy, and C. Zhang, “Exposing attention glitches with flip-flop language modeling,” *arXiv preprint arXiv:2306.00946*, 2023.
- [30] M. M. Olmez, “Pre-survey and post-survey.” <https://t.ly/tdd-surveys>, 2023.
- [31] M. Schäfer, S. Nadi, A. Eghbali, and F. Tip, “An empirical evaluation of using large language models for automated unit test generation,” *IEEE Transactions on Software Engineering*, 2023.
- [32] M. Polo Usaola, I. García-Rodríguez de Guzmán, O. Pedreira, and Á. Saavedra Places, “Automated generation of oracled test cases with regular expressions and combinatorial techniques,” 2020.
- [33] F. A. D. Teixeira and G. Braga e Silva, “Easytest: An approach for automatic test cases generation from uml activity diagrams,” in *Information Technology-New Generations: 14th International Conference on Information Technology*, pp. 411–417, Springer, 2018.
- [34] Z. Xie, Y. Chen, C. Zhi, S. Deng, and J. Yin, “Chatunitest: a chatgpt-based automated unit test generation tool,” *arXiv preprint arXiv:2305.04764*, 2023.
- [35] N. Funabiki, R. Kusaka, N. Ishihara, and W.-C. Kao, “A proposal of test code generation tool for java programming learning assistant system,” in *2017 IEEE 31st International Conference on Advanced Information Networking and Applications (AINA)*, pp. 51–56, IEEE, 2017.
- [36] C. Lemieux, J. P. Inala, S. K. Lahiri, and S. Sen, “Codamosa: Escaping coverage plateaus in test generation with pre-trained large language models,” in *International conference on software engineering (ICSE)*, 2023.
- [37] P. Nie, R. Banerjee, J. J. Li, R. J. Mooney, and M. Gligoric, “Learning deep semantics for test completion,” *arXiv preprint arXiv:2302.10166*, 2023.
- [38] B. Meyer, “Applying ‘design by contract’,” *Computer*, vol. 25, no. 10, pp. 40–51, 1992.

<sup>1</sup><https://github.com/test-skeleton-generator/test-skeleton-generator>

<sup>2</sup><https://zenodo.org/records/10389418>

# Leveraging Open Source LLMs for Software Engineering Education and Training

Juanan Pereira, Juan-Miguel López, Xabier Garmendia, Maider Azanza

*Dept. of Computer Languages and Systems*

*University of the Basque Country, UPV/EHU*

Donostia, Spain

{juanan.pereira, juanmiguel.lopez, xabier.garmendiad, maider.azanza}@ehu.eus

**Abstract**—Generative AI, particularly Large Language Models (LLMs), presents innovative opportunities to enhance software engineering education. Open source LLMs such as LLaMA and Mistral leverage the potential of generative AI offering distinct advantages over proprietary options including transparency, customizability, collaboration, and cost savings. This paper develops a catalog of LLM prompt examples tailored for software engineering training, mapped to knowledge areas from the Software Engineering Body of Knowledge (SWEBoK) framework. Example prompts demonstrate LLMs' capabilities in eliciting requirements, diagram generation, API simulation, effort estimation through role-playing, and other areas. The methodology involves evaluating prompt responses from ChatGPT, Mistral, and LLaMA on representative tasks. Quantitative and qualitative analysis assesses quality, usefulness, and correctness. Findings show ChatGPT and Mistral outperforming LLaMA overall, but no model perfectly executes complex interactions. We examine implications and challenges of integrating open source LLMs into classrooms, emphasizing the need for oversight, verification, and prompt design aligned with pedagogical objectives.

**Index Terms**—Software Engineering Education, Open Source AI Models, Large Language Models, Prompt Engineering

## I. INTRODUCTION

Software Engineering (SE) is a dynamic and intricate discipline that continually evolves, requiring constant learning and adaptation to remain at the forefront of technological advancements. The application of Generative Artificial Intelligence (GenAI) through its application in Large Language Models (LLMs) to SE tasks presents opportunities through the automation of activities like coding, design, and documentation, but also risks around reliability and effectiveness that have broader implications needing consideration [1].

The adaptability and potential of LLMs in general, lie in their ability to encourage motivation, engagement, and learner autonomy by providing personalized, interactive support, thereby enhancing the learning experience [2].

LLM-assisted software engineering promises to support developers with expert knowledge over the whole software lifecycle in a conversational way [3].

However, it is important to note that the quality of LLM output is currently heavily dependent on the quality and scope of their training data, which in the case of software engineering area is still under development.

In the context of SE, LLMs are increasingly utilized for tasks such as code explanation, [4], requirements information

retrieval [5], identifying failure-inducing test cases [6], program repair [7] and vulnerability detection [8].

For instance, it has been demonstrated that prompt design techniques for SE in the form of patterns can be applied to improve requirements elicitation, rapid prototyping, code quality, refactoring, and system design [9]. Moreover, automatic test generation from general bug reports has also been explored using LLMs [10].

Open research challenges around LLMs and SE lie in ensuring the reliability, efficiency and effectiveness of LLMs for automating activities like coding, design, requirements specification, repair, refactoring, optimization, documentation and analytics, given LLMs' tendency to hallucinate solutions [11].

With regard to the educational context, introducing SE concepts to students presents a notable challenge for educators, as highlighted in various studies [12], [13]. The field is in a perpetual state of flux, characterized by the frequent emergence of new programming languages, tools, and methodologies [14]. This constant evolution, coupled with the diverse backgrounds and experience levels of students, makes traditional teaching methods like lectures and lab exercises less effective [15].

In response, there has been a noticeable transition over the past few years from teacher-focused learning methods to student-centered ones [16]. This shift emphasizes the need for innovative technology to facilitate an effective learning process [17].

Other problems encountered in teaching SE were related to the course contents [18]. In this context, the Software Engineering Body of Knowledge (SWEBoKv3) guide [19] serves as an essential framework, outlining the key knowledge areas (KAs) in SE education<sup>1</sup>. Just as LLMs have the potential to support software engineering, they also hold significant promise for learning it, by offering innovative solutions to improve the teaching and learning experience in these KAs. GenAI, particularly through its application in LLMs such as ChatGPT, Anthropic Claude, and Google Bard, presents a novel way to interactively engage students in the software engineering curriculum. These models use advanced machine

<sup>1</sup>It is worth noting that an updated version of the SWEBoK guide (2024 edition) will be released soon, expanding the number of KAs to 18. While this update does not directly affect the findings of this paper, it highlights the evolving nature of the field.

learning algorithms to understand and generate human-like text, providing a unique tool for educators to create dynamic and responsive teaching materials [20].

In this paper, we delve into the utilization of GenAI within the context of SE education, aligning with the KAs defined by SWEBoK. Our approach focuses on developing a set of prompt examples specifically designed for software engineering education using LLMs. We are particularly keen on exploring and evaluating the effectiveness of open-source LLMs like LLaMA or Mistral to take advantage of their transparency, customizability, collaboration, or cost savings aspects. Therefore, we compare their performance and suitability for educational purposes against a proprietary model, GPT-3.5 (the only model available for free to students at the time of this writing). By doing so, we hope to offer insights into how different LLMs can be utilized in software engineering training.

Despite the promising potential of GenAI in education, we also address the challenges and limitations inherent in its application, such as concerns over code quality, overreliance on automation, and the need for thoughtful integration within teaching strategies. By examining recent research and case studies, this paper aims to offer an overview of GenAI's role in SE education, providing a foundation for future research and application in this rapidly evolving field.

On balance, we contribute by:

- exploring the benefits of open source LLMs in SE education (Section III),
- specifying specific examples showing how to use GenAI prompts for teaching ideas in each SWEBOk KA (Table I)
- evaluating the responses from models like ChatGPT, Mistral, and LLaMA on representative tasks in software engineering (Section VI),
- discussing on the potential and challenges of integrating LLMs into software engineering education (Section VII).

This paper presents an initial exploration into the potential of open-source LLMs for software engineering education, examining their effectiveness across various SE tasks and reflecting on the benefits and challenges of integrating them into classrooms. The paper is structured as follows: Section 2 provides a background on Generative AI and open-source LLMs. Section 3 discusses the benefits of open-source LLMs in the context of SE education. Section 4 explores the use of LLMs in the software engineering classroom, and Section 5 outlines our methodology. Section 6 presents our evaluation results, and Section 7 discusses the implications and challenges of our findings. Finally, Section 8 offers concluding remarks and directions for future work in this area.

We start introducing generative AI and open source LLMs.

## II. A BRIEF ON GENERATIVE AI AND OPEN SOURCE LLMs

GenAI is at the forefront of creating new, original content through artificial intelligence, utilizing machine learning algorithms to produce diverse forms of data such as text, images,

music, and videos. These models are capable of recognizing complex patterns and learning from extensive datasets. Their realistic and intricate outputs mimic human creativity, finding practical applications in various fields such as medicine [21] or software development [1]. In the realm of text generation, open-source LLMs like Mistral and LLaMA have garnered attention.

### A. *Mixtral and MistralAI*

MistralAI is an artificial intelligence company focused on developing large language models that can understand and generate natural language. Their latest model, Mixtral-8x-7B, introduces a new Sparse Mixture-of-Experts (SMoE) architecture that achieves state-of-the-art performance while being more efficient.

Specifically, Mixtral is composed of multiple “expert” modules that can each process a subset of the input data. A router mechanism dynamically selects which few experts process each part of the input. This means that even though Mixtral has a very high overall capacity with 47 billion parameters, each part of the input only uses 13 billion active parameters.

Through this efficient architecture, Mixtral matches or exceeds previous state-of-the-art models like GPT-3.5 and LLaMA 2 despite having far fewer parameters. It demonstrates strong performance across diverse Natural Language Processing (NLP) tasks including mathematics, code generation, reading comprehension, common sense reasoning, and multilingual understanding.

While Mixtral-8x7B is one of the best open source LLM, its enormous size (46.7B parameters) means that the model cannot be fully loaded on a consumer GPU (e.g., a nVidia RTX 3090 GPU with 24 GB of VRAM is not enough), even when quantized to 4 bits. In the context of LLMs, quantization refers to a process that reduces the precision of the model’s numerical data to make it more efficient for computation and storage. Essentially, it’s a technique to compress the model size without significantly compromising its performance [22].

Even though we have used Mixtral just in a SE training context, by developing Mixtral and releasing both the base and fine-tuned models publicly under an open license, MistralAI aimed to push forward research and development of safe, capable AI that can serve a wide range of uses.

### B. *LLaMA*

Meta AI’s LLaMA (Large Language Model Meta AI) is a family of language models with various sizes, ranging from 7 to 70 billion parameters. Released in 2023, LLaMA models have demonstrated competitive performance on NLP benchmarks. Notably, the 13-billion parameter model outperformed the much larger GPT-3 in many respects. LLaMA’s training datasets are comprehensive, including web pages, open-source code, Wikipedia, and more, ensuring a diverse knowledge base. Meta released LLaMA under a noncommercial license, significantly contributing to AI research and development.

Unlike proprietary models such as GPT, these open-source alternatives offer a unique set of advantages in terms of

privacy, cost, transparency, community-driven development, customization, avoiding vendor dependency, and educational value. The next section explores the potential of these benefits for SE education.

### III. BENEFITS OF OPEN SOURCE LLMs

The integration of open-source LLMs into SE education presents a realm of untapped potential, offering distinct advantages over their proprietary counterparts. These open-source models not only democratize access to advanced AI technology but also address several concerns associated with proprietary LLMs, such as privacy, compliance with regulations like GDPR, and financial costs.

**Transparency:** Open-source LLMs provide a transparent and customizable platform for educators and students. Unlike proprietary models, where the internal workings are often opaque, open-source models allow SE educators to understand, modify, and adapt the algorithms to suit specific educational needs [23]. This transparency is crucial in a learning environment, as it enables educators to tailor the models to deliver more relevant and effective educational content. Furthermore, the ability to inspect and modify the codebase of open-source LLMs encourages a deeper understanding of the technology, fostering an environment of learning and experimentation that is vital in SE education [24].

**Avoiding Vendor Lock-in:** Using an open source LLM avoids vendor lock-in, where users are dependent on a specific provider's tools and services. This freedom allows for more flexibility in changing or upgrading systems as needed.

**Privacy:** Privacy and data protection are significant concerns with proprietary LLMs, especially in regions governed by strict data protection laws like the GDPR [25]. Open-source LLMs offer greater control over data, allowing institutions to host these models on their servers, thus ensuring that sensitive information, such as student data, remains within the control of the educational institution. This aspect is particularly relevant in scenarios where data privacy is paramount.

**Financial costs:** Economically, open-source LLMs present a more viable option for educational institutions. Proprietary models often come with licensing fees and usage costs, which can be prohibitive for many schools and universities, especially those with limited budgets. Open-source software, including Open-source LLMs, offers a cost-effective alternative to proprietary software in education. Being free to use and modify, alleviates these financial burdens, making advanced AI technology more accessible to a broader range of educational institutions [26].

In contrast, proprietary LLMs, while often more polished and user-friendly, come with the caveat of higher costs, potential privacy concerns, and a lack of transparency that can impede their full utilization in educational settings. These models, bound by company policies and profit motives, may not always align with the educational goals of SE programs.

Seminal works, such as those by White et al. [9], highlight the importance of carefully crafted prompts and strategies in leveraging LLMs effectively in SE. These models offer

innovative solutions for common challenges in SE activities. The next section explores the use of the LLMs for activities conducted during in the Software Engineering Classroom.

### IV. LLMs IN THE SOFTWARE ENGINEERING CLASSROOM

In January 2023, the authors of this manuscript started teaching a new edition of the Software Engineering (SoftEng) undergraduate course with second-year students of the Computer Science Degree at the University of the Basque Country (UPV/EHU). During the course, we wanted to investigate and test the capabilities of ChatGPT for its use in the SoftEng classroom (later, when LLaMA2 and Mistral AI released their models under and Open Source license, we tested them for this paper). ChatGPT chatbot was published by OpenAI just two months before our course started, in November 2022.

The students of the SoftEng course must complete a practical project within 4 months. Starting from a statement, they perform the analysis, design, implementation and testing of a desktop application developed in Java and JavaFX. In 2023 the students, grouped in teams of 4 people, were asked to develop a Mastodon desktop client. Briefly, Mastodon can be described as a decentralized alternative to X (formerly known as Twitter). One of the assignments to accomplish by the students in the course is the requirements analysis and use case gathering: **identifying the functional and non-functional requirements to be implemented**. Given their limited experience, students often struggle with this. They face a blank sheet of paper on which they need to write a list of the functionalities the application should offer. This is where ChatGPT, Mistral, LLaMA and other LLMs can help by creating a first draft to help overcome this first stumbling block.

**Brainstorming.** The following prompt generated a list of functional and non-functional requirements, which served to promote a brainstorming of ideas to be developed among team members during the first iteration.

“A student has been asked to produce a list of use cases for an application that she must develop. The application is a Mastodon desktop client that must be implemented in Java and JavaFX. The student has drafted the following list of use cases:”

**Diagram generation.** The above requirements capture phase is usually accompanied by two UML artifacts: a domain model and a use case diagram. The analyzed chatbots (ChatGPT, LLaMA, Mixtral) do not generate graphs or diagrams directly without the help of plugins, but we can use a detour to generate UML diagrams. Just asking the chatbot to output its response in PlantUML format does the trick. PlantUML<sup>2</sup> is a tool to quickly generate UML diagrams from plain text. The following prompt allowed to generate a domain model of the entities and relationships existing in the Mastodon client project:

“Identify the entities and relationships of the mastodon client project. Give your answer in PlantUML format”

<sup>2</sup><https://plantuml.com/>

The answer in PlantUML format can be pasted into an online PlantUML viewer (such as planttext.com) to quickly view the results.

**Simulation.** Among the suggested functional requirements was the Timeline View feature. The user should be able to view their home timeline, which includes the posts of the users they follow, in chronological order, within the desktop client. To implement this functionality - and others - students wanted to use the Mastodon API. However, in the early stages of the project there is no information yet on how to interact with this API. It is necessary to simulate it. This is another advantage offered by ChatGPT and similar generative AIs. A GenAI can effectively simulate an API and guide the student to make good design decisions that result in a robust, scalable and usable application. The interactive nature of working with GenAI also provides a valuable learning experience for students. See <https://sharegpt.com/c/Ik2FYBZ5> for an example where the student can easily ask the chatbot to simulate a REST API (not yet implemented) with a simple line. The following prompt gets a response from ChatGPT where it returns a JSON object that we could receive if we called a hypothetical api/timeline method of the Mastodon API: "Can you generate a draft HTTP API for the timeline view?"

**Role playing estimations.** Another area of a software project where students struggle due to the complexity of the topic and their inexperience is effort estimation. Again, GenAI can help to improve this skill by following the technique of role-playing estimations. With this technique we can ask the IA to ask the learner to estimate the effort required to complete certain functionalities. The IA will then respond, as a counsellor, by giving feedback on the estimate. The following prompt is an example of this technique:

"I want you to ask me about estimating 3 use cases, one at a time. For each one, I will answer you with a rough estimate in hours. Then you have to give me feedback (with your opinion on my estimate for each use case)" (Fig. 2).

This role playing usage of the AI is interesting, not only because of the immediate feedback, but also because lecturers can use it to measure the accuracy of students' relative estimates (according to the student, is task A more or less difficult to implement than task B, and why is this so?).

Based on our experience we collected a set of useful prompts - like the ones described above - and categorized them in 12 knowledge areas (KAs), following the classification defined by the SWEBOK guideline (see Table 1). For each KA we added one or two specific examples, describing the type of activity (role playing, simulation, brainstorm, etc.) and providing a specific example of a prompt and its response in ChatGPT, Mistral8x7b and LLama2. Any interested educator can follow that table and get some actionable ideas to apply in their classroom.

## V. METHODOLOGY

Following the scenario described in Section IV for the Mastodon Application Design Project, we asked GPT, LLAMA

and Mixtral models to answer to the same 4 prompt examples, highlighted in gray in Table I:

- Brainstorming (generate a list of functional and non-functional requirements)
- Diagram generation: generate the domain model diagram (entities and relationships)
- Simulation: ask to simulate an API endpoint
- Role Playing estimation: ask the user to estimate the effort in hours of 3 use cases and give feedback.

The examples cover 3 different KAs: Software Requirements, Design, Testing and Maintenance. For each prompt we generated responses from 3 different LLMs, using the API provided by MistralAI for Mixtral8x7b (medium), the API provided by Together.AI for NousHemes/LLaMA2-70b, and the API provided by OpenAI for GPT 3.5. We chose the 3.5 version of GPT due to its free cost (so we can compare it with the free and open source alternatives, instead of using GPT4).

The system prompt for the Open Source models was: "You are a helpful software engineering assistant".

We ran the queries in a custom web application using LLMs APIs, Python and Streamlit [27] (see Fig. 4). We also developed a custom browser extension to save the results of the conversations in a database so we can export them to external systems (in our case, to Google Drive). We stored the conversations in three web anonymous Drive documents, for Mixtral<sup>3</sup>, LLaMA<sup>4</sup>, and GPT-3.5<sup>5</sup>. Then, the manuscript authors (software engineering teaching colleagues) evaluated the responses, comparing quality, usefulness and correctness across models. We also used two state of the art LLMs (GPT4 and Claude) as external evaluators or judges. To avoid biased responses (for instance, to avoid GPT4 to recognize the answers from GPT3.5) we removed any reference to the name of the LLMs in the responses to be evaluated. We uploaded three text files to each LLM-judge and used the following prompt to obtain their assessments:

*You are a software engineering teacher that needs to evaluate a set of answers to software engineering questions related to the analysis, design and implementation of a Mastodon client generated by 3 Large Language Models. I have attached three text files with the answers of LLM1, LLM2 and LLM3. Read the questions (they are all the same for the three LLMs) and the answers (each file's name is linked to each LLM: llm1.txt, llm2.txt, llm3.txt). Please, evaluate the answers of each LLM regarding the quality, usefulness (how useful could the answers be for a software engineering student) and correctness. Please, the evaluation has to include a score from 1 to 5 and explain the rationale behind your answers.*

## VI. RESULTS

The evaluation aims to examine the capabilities of different LLMs on SE tasks through both quantitative model evaluation

<sup>3</sup><https://tinyurl.com/mixtral-medium>

<sup>4</sup><https://tinyurl.com/llama2-70b>

<sup>5</sup><https://tinyurl.com/GPTv3-5>

TABLE I

A SUMMARY TABLE OF SPECIFIC EXAMPLES SHOWING HOW TO USE GENAI PROMPTS FOR TEACHING IDEAS IN EACH SWEBOK KA. THE THIRD COLUMN IS A LINK TO A CONVERSATION SAVED IN SHAREGPT.COM. IT ONLY SHOWS THE #ID OF THE CONVERSATION (USE THE LINK <HTTP://TINYURL.COM/#ID> TO OPEN THE LINK BY CLICKING ON IT. LINKS WITH THE SAME ID ARE PART OF A SINGLE CONVERSATION WITH THE LLM WHERE DIFFERENT PROMPTS WHERE PUT TO THE TEST. HIGHLIGHTED LINES ARE PROMPTS USED IN CHATGPT, MISTRAL AND LLAMA FOR COMPARISON.

Knowledge Area		Example <a href="https://tinyurl.com/#ID">https://tinyurl.com/#ID</a>
<b>Software Requirements</b>		
Brainstorm		
UML Diagrams	Offer open questions for thinking about an issue	ypz4c53m
Review	Generate requirements diagrams in UML	ad4seeux
	Give feedback to the student about the quality of their requirements	49pzep2r
<b>Software Design</b>		
Suggest best practices	Given a software design, offer suggestions for refactoring (e.g. for applying design patterns)	3vznppc
Simulation	Simulate how a system would work before implementing it	4f8ae2pd
<b>Software Construction</b>		
Provide challenges	Provide an incomplete solution to a problem and ask/challenge the student to fill-in the gaps.	e5v7fys4
Generate new educational material	Self-taught students or learners with a desire to learn more can ask the GenAI to create new explanations for unknown or not well-known material.	y99k7px7
<b>Software Testing</b>		
Bootstrap testing code	Ask the students to generate testing code for a method. Then ask GenAI to do the same. Compare both solutions.	3xt832d4
<b>Software Maintenance</b>		
Detect bugs	Show bugged source code to students and to GenAI. Ask them to detect the bug and to explain how to solve it. Compare solutions.	cOF9ak8
Discuss source code	Discuss source code from the point of view of maintenance	2bc4ks7f
<b>Software Configuration Management</b>		
Automation & Scaffolding	Generate software configuration management and discuss the solution in class	48r7sv45
<b>Software Engineering Management</b>		
Help students to estimate	Estimating effort and cost is a difficult task for any engineer, and even more so for students with little experience.	585hzvkz
<b>Software Engineering Process</b>		
Provide auto evaluation quizzes	Given the following text, create a quiz of 5 questions. Each question will have 4 possible answers from which only one will be correct. Explain the correct solution. Here is the text: [provide the text]	2znajakj
Help students to understand complex topics	Gradually adjusting the difficulty of the explanation "explain it to a 13 years old student"	2br9bdzm
<b>Software Engineering Models and Methods</b>		
Suggest an agenda of questions	GenAI can help the scrum master (a role played by a student) to prepare daily scrum meetings in advance	5n8zra6s
<b>Software Quality</b>		
Answer software quality related questions	E.g. "Can you give me the cyclomatic complexity value of this example() method?"	IAtTqpH
Detect and refactor code smells	Explaining to the students, with specific examples, how their code smells affect software quality	mc5zxy7j
<b>Software Engineering Professional Practice</b>		
Stimulate discussions on related topics	E.g. suggesting open questions about legal issues in software	mrx8czap
<b>Software Engineering Economics</b>		
Role playing estimations	Ask about three use cases for the student to estimate in hours, then provide feedback on each estimation.	585hzvkz

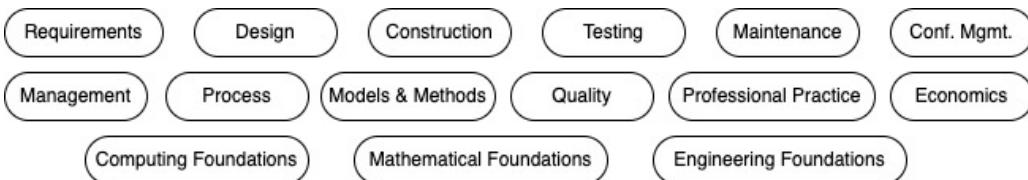


Fig. 1. SWEBOK defines 15 KAs related to software engineering. For this work, we are interested in the ones shown in figure but not in computing, mathematical and engineering foundations .

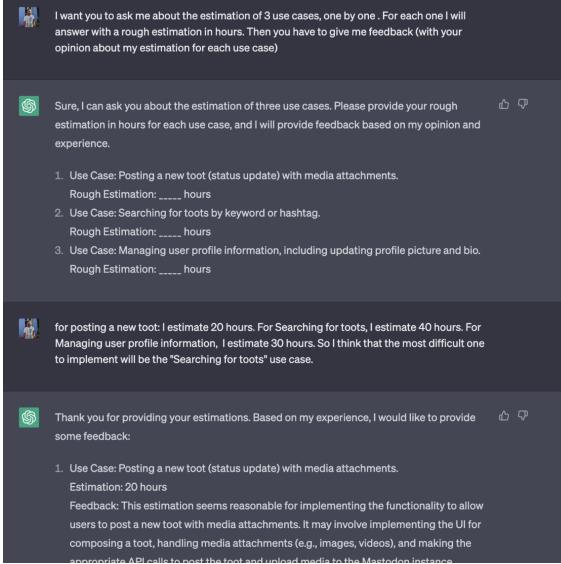


Fig. 2. GenAI can play the role of an external evaluator of students' goals, estimates and plans

TABLE II

**AUTHORS EVALUATION OF THE LLM RESPONSES, COMPARING QUALITY, USEFULNESS AND CORRECTNESS ACROSS MODELS (LIKERT SCALE FROM 0 - VERY POOR - TO 5 - EXCELLENT -**

Model	Quality	Usefulness	Correctness
LLaMA2-70b	2	3	2
Mixtral-8x7b	5	4	4
GPT3.5	4	4	4

and qualitative assessment. The quantitative model evaluation provides a numeric scoring of the models' outputs on key criteria relevant for SE tasks. The qualitative assessment provides a more in-depth look at differences between the LLMs' abilities to elicit use case requirements, role play the estimation of tasks, simulate an API endpoint and generate a Domain Model diagram. When combined, the quantitative and qualitative assessments provide a multifaceted understanding of the models' capabilities.

#### A. Model evaluation

Table II presents the authors' (human) evaluation of the quality, usefulness, and correctness of responses from different LLMs on a 5-point Likert scale. Tables III and IV show the evaluation provided by the LLM-judges.

Regarding the human evaluation, obtained by consensus after discussion, the GPT-3.5 model received high scores of 4 out of 5 on all three metrics. Mixtral-8x7b performed even better on quality with a perfect score of 5, and scored 4 on usefulness and correctness. In contrast, LLaMA2-70b received lower scores, with a 2 out of 5 on both quality and correctness, and a 3 on usefulness. Overall, the results show GPT-3.5 and Mixtral-8x7b generating higher quality, more useful, and more correct responses compared to LLaMA2-70b.

TABLE III

**GPT4 EVALUATION OF THE LLM RESPONSES, COMPARING QUALITY, USEFULNESS AND CORRECTNESS ACROSS MODELS (LIKERT SCALE FROM 0 - VERY POOR - TO 5 - EXCELLENT -**

Model	Quality	Usefulness	Correctness
LLaMA2-70b	4	4	4
Mixtral-8x7b	5	5	5
GPT3.5	4	4	4

TABLE IV

**CLAUDE EVALUATION OF THE LLM RESPONSES, COMPARING QUALITY, USEFULNESS AND CORRECTNESS ACROSS MODELS (LIKERT SCALE FROM 0 - VERY POOR - TO 5 - EXCELLENT -**

Model	Quality	Usefulness	Correctness
LLaMA2-70b	3	4	4
Mixtral-8x7b	4	5	4
GPT3.5	5	5	5

Authors' opinions are aligned with GPT4's evaluation (Table III). GPT4 rationale was the following (the exact names of each LLM have been added by the authors, in parenthesis; as explained the LLM-judges did a blind evaluation):

"LLM1 (LLaMA2-70b), LLM2 (Mixtral), and LLM3 (GPT3.5) provided high-quality, useful, and technically correct answers to the software engineering questions posed. LLM2 (Mixtral) stands out slightly in terms of quality and depth, particularly in the API draft and the detailed feedback on estimations. All models would serve as valuable educational tools, offering students practical examples of how to approach software engineering tasks."

Regarding Claude's assessment in Table IV, the judge assigned high grades to both Mixtral and GPT-3.5. However, Claude preferred the answers from GPT-3.5 over Mixtral's.

Claude's rationale for its answer was the following:

"“LLM2 (Mixtral) and LLM3 (GPT3.5) provided the most comprehensive, technically sound, and useful examples for a software engineering student. Their understanding of software requirements, API design, and system modeling indicates strong technical knowledge. LLM1's (LLaMA2-70b) examples, while decent, lacked some of the rigor and polish of the other two. Overall, LLM2 and LLM3 demonstrate the most mastery.”"

#### B. Qualitative assessment

We will split our assessment in 4 parts, one for each prompt:  
**Models' ability to elicit use case requirements**

ChatGPT provided the most comprehensive list of use cases covering major functionalities. It has detailed and appropriately named use cases like "Posting Toots" or "Timeline Viewing" and explained purpose and expected behavior for some use cases. This shows an understanding of identifying core software requirements and articulating use cases.

Mixtral also produced a reasonably thorough list of use cases with names that clearly explain intended functions. It has also grouped related cases (e.g. User Profile Management).

Fig. 3. Mixtral was the only LLM capable of generating a proper Domain Model diagram in PlantUML. GPT35 failed to generate a syntactically correct diagram so the generation failed. LLaMA2 attempt was very poor - although the diagram was syntactically correct.

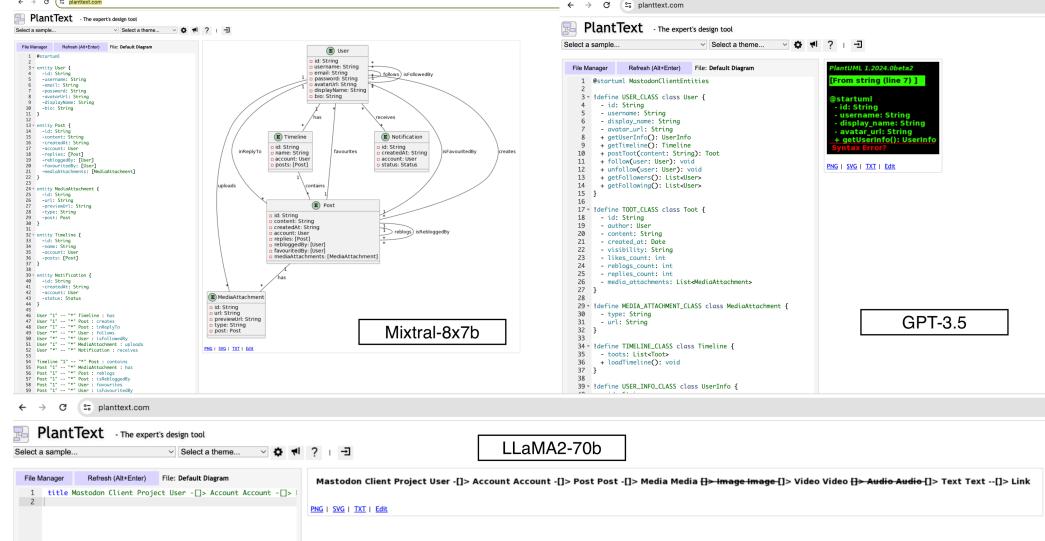
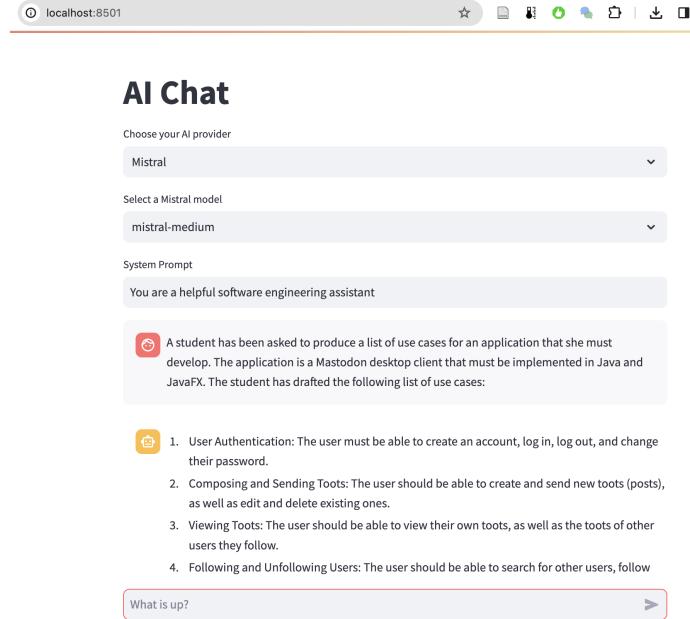


Fig. 4. We implemented a custom web interface using Streamlit for development and testing of Open Source LLMs. The conversations can be saved using a custom browser extension developed for this article.



While not as extensive as ChatGPT's, Mixtral generates a solid set of baseline use cases.

LLaMA2 starts with a very basic list of use cases that lacks the breadth or description in the other LLMs. LLaMA2 clearly has the right methodology but weaker content generation than ChatGPT or Mixtral. Those two models demonstrate greater mastery of eliciting use case requirements.

**Models' ability to role play the estimation of tasks**  
ChatGPT makes a reasonable attempt at asking the student to estimate 3 use cases and provide feedback. However, provided feedback is very generic - simply stating the estimate seems reasonable without specifics and ends after just the first role

turn (even if we asked to play the role-playing game with 3 uses cases).

Mixtral has a good realistic and helpful approach to estimations. It provides context and considerations to factor in for each use case. However, it fails to play the role-playing turn based schema (it asks and answers its own questions, simulating feedback)

LLaMA2 completely hallucinates the use cases and does not understand the role playing game, either (it both asks and answers its own questions).

**Models' ability to simulate an API endpoint.** ChatGPT provides a reasonable draft GET endpoint for the Timeline

view, including expected parameters like post IDs and pagination. It also outlines request structure, response payload and codes, covering the key aspects required in an API specification, showing competence.

Mixtral generates a GET endpoint for Timeline view with an example payload. It also calls out the need to adapt to project's specific requirements and notes the lack of error handling in the draft. Mixtral also defines an appropriately scoped endpoint while acknowledging further enhancements needed.

LLAMA2, gives very basic CRUD endpoints for Timeline without including any payload structure or examples. Its response is a very basic one compared to the other two models.

Thus, ChatGPT and Mixtral produce more realistic draft endpoints with details compared to LLAMA2's cursory CRUD resources. Mixtral's self-awareness of the draft limitations is also noteworthy. Overall, ChatGPT and Mixtral better demonstrate the knowledge and communication for simulating API specification tasks.

**Models' ability to generate a Domain Model diagram.**  
Mixtral is the only model capable of providing a complete view of a UML diagram without syntax errors in the diagram code. The domain model includes various entities such as User, Post, MediaAttachment, Timeline, and Notification with attributes and relationships detailed, like "Users can follow each other", "Posts can contain MediaAttachments", "Posts can be reblogged or favorited by Users".

ChatGPT's attempt produces an incorrect PlantUML diagram that cannot be rendered. The diagram tries to include class definitions for User, Toots (messages), MediaAttachment, and Timeline, with relationships between them. But as indicated, the output cannot even be redisplayed.

LLama's attempt is a very poor solution. It can be rendered but its quality and semantics are close to zero.

## VII. DISCUSSION

Overall, ChatGPT and Mixtral produced higher quality, more useful, and more correct responses compared to LLAMA2. ChatGPT received scores of 4 out of 5 on all metrics while Mixtral received scores of 5, 4, and 4. In contrast, LLAMA2's scores were just 2, 3, and 2. ChatGPT and Mixtral consistently outperformed LLAMA2 across the evaluation prompts, demonstrating stronger capabilities for SE education tasks. But no model perfectly executed certain interactions like role playing. Although it could be due to a lack of proper prompts. The results of this analysis highlight several key implications for integrating open source LLMs like LLAMA and Mistral into software engineering classrooms:

- Open source LLMs can help students overcome initial barriers when faced with open-ended tasks like requirements gathering or estimations. The brainstorming and role playing capabilities of models like ChatGPT and Mistral demonstrate their potential to provide scaffolds to support student learning.
- Transparency of open source LLMs allows for customization and alignment with curriculum goals. Unlike black-box proprietary models, open source LLMs can be

inspected, adapted, and tuned to suit specific educational needs. This is especially relevant in specialized domains like SE.

- Running costs can be lower with local deployment options for open source models, making them accessible to more institutions. Smaller schools may lack resources for expensive proprietary services.

Although there is potential for improving software engineering education with open source LLMs, it is still important to integrate them responsibly. To ensure that there is no over-reliance on model outputs, educators must oversee and verify the results in order to look for any potential errors. It might be necessary to realign curricula and assessments in order to use LLMs to enhance rather than substitute human teaching and learning.

Infrastructure requirements like GPUs for running certain models can also incur high upfront costs for institutions. Rather than acquiring high-end GPU hardware, an alternative is to utilize inference providers to rent running time for LLMs. For this work, we used the together.ai API to run inferences on LLAMA2-70b, since we lacked local GPUs. Pricing varies by provider; at together.ai LLAMA2 costs \$0.9 per million tokens.

We elected to use the Mistral API for a clear cost structure, with Mistral-medium priced at €2.5 per million input tokens and €7.5 per million outputs. In our case, cloud-based inference enabled flexible LLM experimentation without large up-front infrastructure investments. Nevertheless, as the models advance, open source LLMs show promise as an addition to revolutionize personalized, interactive education at scale when utilized sparingly as opposed to viewing outputs as gospel. All in all, open source LLMs present valuable opportunities but require prudent integration and moderation to address ongoing challenges around hallucination and misinformation.

While LLMs offer promising opportunities, it is essential to acknowledge potential risks associated with their use in SE education. One major concern is **hallucination**, where LLMs can generate plausible-sounding but incorrect or nonsensical information, potentially misleading students. Furthermore, **biases** present in the training data can be reflected in LLM outputs, perpetuating societal biases or unfair practices. Another risk is **over-reliance on automation**, where students might become overly dependent on LLMs for generating solutions, potentially hindering the development of critical problem-solving and coding skills. Finally, **ethical considerations** arise regarding plagiarism, as students might be tempted to present LLM-generated code as their own, and intellectual property concerns might arise if LLM-generated code infringes on existing copyrights. Therefore, integrating LLMs into SE education requires careful consideration of these risks, emphasizing critical thinking, ethical practices, and the importance of validating LLM outputs.

## VIII. CONCLUSIONS

In examining the role of open source LLMs in SE education, several important insights emerge:

**Promising Educational Tools.** Models like ChatGPT and Mistral are promising in enhancing student motivation and aiding in learning through interactive prompts. They have shown effectiveness in complex tasks such as eliciting requirements, making estimations, and drafting specifications.

**Need for Oversight and Verification.** While these models are useful, they are not infallible. Oversight is essential, especially in complex interactions, to ensure their outputs align with educational objectives and require validation before being integrated into the classroom.

**Guidance from Established Frameworks.** Frameworks like SWEBoK can inform the development of comprehensive benchmarks for prompts, specifically tailored for SE. Standardized datasets are also crucial for rigorous testing and evaluation of LLMs.

**Balancing Transparency and Costs.** The open-source nature of these LLMs offers opportunities for customization according to educational contexts. However, the associated infrastructure costs could be a barrier for smaller institutions. Solutions like grants or shared resources could help mitigate this.

**Importance of Thoughtful Prompt Design.** The effectiveness of large language models greatly depends on the design of prompts. These should be aligned with curriculum goals and if possible, follow a prompt catalog of good examples that help obtain high-quality outputs.

Studying different models in software engineering (SE) scenarios can help us learn how to better create prompts and complete tasks. As new versions of large language models (LLMs) come out, we can update our information to show their improved abilities in SE challenges. Regularly checking the latest models will show us how fast LLMs are getting better at important skills needed for SE work. By comparing different LLMs, both open source and proprietary, we can find the best ways to use these models as helpers. Open source LLMs could also make software engineering education better by offering lessons that adjust to each learner. With effective evaluation methods and teamwork, Generative AI models could change the way we teach and learn technical skills.

While our initial evaluation shows the strengths of these models in handling complex SE tasks, it is important to note that this is just the first step in a longer research path. Our future work will focus on improving the outputs of these models by fine-tuning them on SE-specific datasets (previous research [28] demonstrated LLMs trained using domain-focused corpora perform better on specialized tasks) and improving our prompt design strategies, conducting studies with our students, and including new open-source LLMs in our project.

- **Fine-tuning and Prompt Engineering:** We recognize that fine-tuning these LLMs on SE-specific datasets, along with careful prompt design, is likely to bring further improvements in their performance and educational value. Our future work will focus on developing and testing fine-tuned models tailored to specific SWEBOk KAs. We will also look more into prompt design, exploring ways to

create prompts that draw out more targeted and useful responses from the LLMs.

- **Empirical Studies with Students:** To fully understand the impact of open-source LLMs on SE education, we plan to conduct empirical studies involving students using these models as learning tools. These studies will allow us to collect data on student performance, engagement, and learning outcomes, providing concrete evidence of the educational benefits of open-source LLMs.

- **Expanding the Scope:** As the field of open-source LLMs rapidly evolves, we intend to broaden our scope to include newer models as they become available. This will provide a more comprehensive understanding of the strengths and weaknesses of different models in the context of SE education.

#### ACKNOWLEDGMENT

This research is supported by contract GIU21/037 under the program "Convocatoria para la Concesión de Ayudas a los Grupos de Investigación en la Universidad del País Vasco/Euskal Herriko Unibertsitatea (2021)." Additional support is provided by MCIN/AEI/10.13039/501100011033 and the European Union NextGeneration EU/PRTR under contract PID2021-125438OB-I00. Funding was also received from the University of the Basque Country (UPV/EHU) under the "University-Enterprise-Society" program (US22/17).

## REFERENCES

- [1] I. Ozkaya, "Application of Large Language Models to Software Engineering Tasks: Opportunities, risks, and implications," *IEEE Software*, vol. 40, no. 3, pp. 4–8, 2023.
- [2] M. Firat, "How ChatGPT can transform autodidactic experiences and open education," *Department of Distance Education, Open Education Faculty, Anadolu Univ*, 2023.
- [3] L. Belzner, T. Gabor, and M. Wirsing, "Large Language Model Assisted Software Engineering: Prospects, challenges, and a case study," in *Bridging the Gap Between AI and Reality*, B. Steffen, Ed. Cham: Springer Nature Switzerland, 2024, pp. 355–374.
- [4] E. Chen, R. Huang, H.-S. Chen, Y.-H. Tseng, and L.-Y. Li, "GPTutor: a ChatGPT-powered programming tool for code explanation," 2023.
- [5] J. Zhang, Y. Chen, N. Niu, and C. Liu, "A preliminary evaluation of ChatGPT in requirements information retrieval," 2023.
- [6] T.-O. Li, W. Zong, Y. Wang, H. Tian, Y. Wang, S.-C. Cheung, and J. Kramer, "Finding failure-inducing test cases with chatgpt," 2023.
- [7] J. Cao, M. Li, M. Wen, and S. chi Cheung, "A study on prompt design, advantages and limitations of chatgpt for deep learning program repair," 2023.
- [8] A. Cheshkov, P. Zadorozhny, and R. Levichev, "Evaluation of chatgpt model for vulnerability detection," 2023.
- [9] J. White, Q. Fu, S. Hays, M. Sandborn, C. Olea, H. Gilbert, A. Elnashar, J. Spencer-Smith, and D. C. Schmidt, "A prompt pattern catalog to enhance prompt engineering with chatgpt," 2023.
- [10] S. Kang, J. Yoon, and S. Yoo, "Large language models are few-shot testers: Exploring llm-based general bug reproduction," in *Proceedings of the 45th International Conference on Software Engineering*, ser. ICSE '23. IEEE Press, 2023, p. 2312–2323. [Online]. Available: <https://doi.org/10.1109/ICSE48619.2023.00194>
- [11] A. Fan, B. Gokkaya, M. Harman, M. Lyubarskiy, S. Sengupta, S. Yoo, and J. M. Zhang, "Large language models for software engineering: Survey and open problems," 2023.
- [12] S. Ouhbi and N. Pombo, "Software engineering education: Challenges and perspectives," in *2020 IEEE Global Engineering Education Conference (EDUCON)*. IEEE, 2020, pp. 202–209.
- [13] I. Alves and C. Rocha, "Qualifying software engineers undergraduates in devops - challenges of introducing technical and non-technical concepts in a project-oriented course," *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)*, pp. 144–153, 2021.
- [14] S. Murthy, "Difficulty of Software Engineering and Ways to Overcome Common Challenges," Feb. 2023. [Online]. Available: <https://tinyurl.com/wguedublog>
- [15] O. Shabalina, N. Sadovnikova, and A. Kravets, "Methodology of teaching software engineering: Game-based learning cycle," in *2013 3rd Eastern European Regional Conference on the Engineering of Computer Based Systems*. IEEE, 2013, pp. 113–119.
- [16] N. Battaglia, C. Neil, and M. De Vincenzi, "Software engineering competence-based learning in collaborative virtual environments," in *2021 IEEE World Conference on Engineering Education (EDUNINE)*, 2021, pp. 1–5.
- [17] N. Ngadiman, S. Sulaiman, N. Idris, M. R. Samigan, and H. Mohamed, "Checklist approach for the development of educational applications by novice software developers," *IEEE Access*, vol. 11, pp. 900–918, 2023.
- [18] O. Shata, "Teaching software engineering: Problems and suggestions," *International Journal of Computer Science Issues (IJCSI)*, vol. 8, no. 1, p. 232, 2011.
- [19] IEEE Computer Society, *SWEBOk: Guide to the Software Engineering Body of Knowledge*. Piscataway, NJ: IEEE Press, 2014.
- [20] J. G. Meyer, R. J. Urbanowicz, P. C. Martin, K. O'Connor, R. Li, P.-C. Peng, T. J. Bright, N. Tattonetti, K. J. Won, G. Gonzalez-Hernandez *et al.*, "Chatgpt and Large Language Models in academia: opportunities and challenges," *BioData Mining*, vol. 16, no. 1, p. 20, 2023.
- [21] A. J. Thirunavukarasu, D. S. J. Ting, K. Elangovan, L. Gutierrez, T. F. Tan, and D. S. W. Ting, "Large language models in medicine," *Nature medicine*, vol. 29, no. 8, pp. 1930–1940, 2023.
- [22] HuggingFace. (2023) Quantization. [Online]. Available: [https://huggingface.co/docs/optimum/concept\\_guides/quantization](https://huggingface.co/docs/optimum/concept_guides/quantization)
- [23] Y. Fan, F. Jiang, P. Li, and H. Li, "Grammargpt: Exploring open-source llms for native chinese grammatical error correction with supervised fine-tuning," in *CCF International Conference on Natural Language Processing and Chinese Computing*. Springer, 2023, pp. 69–80.
- [24] M. Müller, C. Schindler, and W. Slany, "Engaging students in open source: Establishing foss development at a university," 2019.
- [25] R. Plant, V. Giuffrida, and D. Gkatzia, "You are what you write: Preserving privacy in the era of large language models," *arXiv preprint arXiv:2204.09391*, 2022.
- [26] A. Khelifi, M. Talib, M. Farouk, and H. Hamam, "Developing an initial open-source platform for the higher education sector—a case study: Alhosn university," *IEEE transactions on learning technologies*, vol. 2, no. 3, pp. 239–248, 2009.
- [27] Streamlit • a faster way to build and share data apps. [Online]. Available: <https://streamlit.io/>
- [28] B. Bhattacharjee, A. Trivedi, M. Muraoka, M. Ramasubramanian, T. Udagawa, I. Gurung, R. Zhang, B. Dandala, R. Ramachandran, M. Maskey, K. Bugbee, M. Little, E. Fancher, L. Sanders, S. Costes, S. Blanco-Cuaresma, K. Lockhart, T. Allen, F. Grezes, M. Ans dell, A. Accomazzi, Y. El-Kurdi, D. Wertheimer, B. Pfitzmann, C. B. Ramis, M. Dolfi, R. T. de Lima, P. Vagenas, S. K. Mukkavilli, P. Staar, S. Vahidinia, R. McGranaghan, A. Mehrabian, and T. Lee, "Indus: Effective and efficient language models for scientific applications," 2024.

# Competence-based Assessment of Programming Assignments

Reinhold Plösch

*Business Informatics - SE*

*Johannes Kepler University Linz*

Linz, Austria

reinhold.ploesch@jku.at

Iris Groher

*Business Informatics - SE*

*Johannes Kepler University Linz*

Linz, Austria

iris.groher@jku.at

Alexander Hofer

*Business Informatics - SE*

*Johannes Kepler University Linz*

Linz, Austria

alexander.hofer@jku.at

**Abstract**—In recent years, computer science education has increasingly focused on the development and application of automated code assessment methods. Test-based assessments of student submissions generate large quantities of data that could be used for learning analytics. Existing literature highlights a predominant use of unit tests for grading rather than for extracting insights into students' challenges in learning programming concepts. Moreover, competence models, which systematically outline required learning outcomes, are increasingly important during curriculum design. We introduce a novel approach for systematically developing programming assignments aligned with competence models. This method also aids educators in creating unit tests that support competence-based learning analytics. As our method maps competences to individual test cases and thereby quantifies a student's proficiency in an assignment, this helps educators to evaluate, whether students have the ability to understand and master the required competences. Experimental application of our method demonstrated enhanced clarity in understanding student assignments and a considerable improvement in the quality of learning analytics data.

**Index Terms**—competence, learning analytics, competence-based learning, automated assessment

## I. INTRODUCTION

In the domain of introductory programming education, novices frequently encounter challenges, often resulting in high drop-out rates [Nikula et al.(2011)], [Bennedsen and Caspersen(2019)]. We also observed this phenomenon in the introductory Java programming course at our institution. Many of the students taking the course are engaging with programming concepts for the first time, and only a few bring prior experience from programming courses taken during their secondary education. The proportion of students failing to pass their exam in our introductory Java programming course has remained stable at around 40%.

Survey data from student feedback indicates a general satisfaction with the programming course. Students commonly identify the intrinsic complexity of the subject matter as a primary source of difficulty, highlighting the challenging aspects of learning programming, especially for beginners. To better support students throughout their learning journey and to improve their performance and reduce dropout rates, our goal was to identify the specific topics within the course that pose the greatest difficulty for students.

To gain a deeper understanding of the challenges faced by our students, we initiated an analysis of their weekly homework assignments. In our course, we employ the Learning Management System (LMS) ArTEMiS [Krusche and Seitz(2018)] which facilitates the use of unit tests to generate an initial grading for exercises. Additionally, it offers immediate feedback to students while they work on their assignments. We conducted an analysis of the unit test results from student submissions across ten assignments of two semesters in the years 2021 and 2022. To get a more comprehensive overview than that provided by individual unit tests, we defined error categories (functional correctness, management of boundary conditions, robustness, and output formatting) and mapped them to all unit tests. Our analysis indicated common difficulties faced by students that increased in assignments dealing with arrays. While this gave us initial hints on topics and associated tasks that might require more instructional focus, we were not able to get a detailed understanding of the students' learning progress and the challenges they faced.

Our primary objective was to identify the programming *competences* that students acquire over the course and to identify the competence areas in which they continue to experience difficulties. Competences specify what the learners will be able to do with the concepts they have learnt [Rugelj et al.(2010)]. Based on the results of our initial unit test-based analysis, we propose a novel Competency Based Learning Analytics (CBLA) method. Our method aims at deriving exercise and programming task specifications that match required competences and to implement autograding support in a way, that not only autograding is supported, but also competence-related learning analytics data can be collected. The competence model we developed for our introductory programming course will be described in Section III-B.

Using our method, we have examined student performance in assignments focusing on arrays from the years 2021 and 2022. The transition to competence-based unit tests enhanced the quality and depth of the learning analytics data we gathered. In particular we saw a clear positive learning trend for some of the competences, while others depend on prerequisite competences, which were, in some cases, inadequately mastered.

The remainder of the paper is structured as follows. Section

2 presents our research goal and the research questions we aim to answer. Section 3 provides some background on the topics of learning analytics and competence-based learning including the competence model we developed for our course. Section 4 discusses work related to our research. In Section 5 we describe the course setting of our introductory programming course including an overview of the initial unit-test based analysis we conducted. Section 6 presents our novel method for competence-based learning analytics. The application of this method to assignments focusing on arrays and the corresponding results are presented in Section 7. Finally, Section 8 concludes the paper and Section 9 discusses potential future research directions.

## II. RESEARCH GOAL AND QUESTIONS

The goal of this paper is to provide a method for systematically developing competence-based unit tests for programming exercises, enhancing our ability to identify specific competences students have difficulties with. The method should further ensure that (1) all competences outlined in the course are addressed through designated exercises; (2) each competence is assessed independently; and (3) the method should provide insightful learning analytics data helping educators in refining their courses.

We define the following three research questions:

- **RQ1:** How can the categorization of error types in ordinary unit tests applied to student submissions in introductory programming courses reveal specific learning difficulties?
- **RQ2:** How can we support the systematic development of competence-based exercises and associated unit tests from a competence model?
- **RQ3:** What is the impact of implementing competence-based exercises and associated unit tests, derived from a competence model, on the quality of the learning analytics data?

To answer RQ1, we will analyse the unit test results of all assignment submissions from the years 2021 and 2022. The test results are obtained from our LMS ArTEMiS. We will answer RQ2 by proposing the CBLA-method for competence-based learning analytics. To evaluate the impact of our proposed method on the quality of the learning analytics data, and to answer RQ3, we apply the proposed model and analyse the results of the competence-based unit tests on assignments focusing on arrays from the years 2021 and 2022.

## III. BACKGROUND

### A. Learning Analytics

Learning analytics, an emergent field, combines methods from diverse disciplines, including machine learning, data science, statistics, psychology, and computer science, applying them in the context of education. It focuses on the measurement, collection, analysis, and reporting of data about learners and their contexts. Due to the widespread use of LMS, large volumes of data are generated, that can be used to derive

insights for improving course quality, student satisfaction, and student performance [Joksimovic et al.(2019)].

Learning analytics distinguishes student-oriented and teacher-oriented learning analytics. The former aims to understand and improve the learning process from the student's perspective, and the latter aims to enhance teaching practice effectiveness.

Student-oriented learning analytics frequently use dashboards, which are often embedded directly within the LMS and automatically generate and update data upon student access [Arnold and Pistilli(2012)], [Krusche and Seitz(2018)]. While such data can be useful, students often feel overwhelmed and are uncertain about how to interpret the information for performance improvement [Arnold and Pistilli(2012)]. Moreover, for students already facing challenges, being confronted with data on their poor performance may potentially trigger a negative feedback loop [Joksimovic et al.(2019)]. More successful implementations of these systems have been observed when educators review the data together with students, providing additional interpretation and guidance [Arnold and Pistilli(2012)].

Teacher-oriented learning analytics primarily consist of predictive models focusing on an early identification of students likely to fail a course and automated reporting on overall student performance [Arnold and Pistilli(2012)], [Olivé et al.(2018)], [Krusche and Seitz(2018)]. These approaches summarise and analyse students' performance and demographic data, enabling educators to proactively provide targeted support to address potential misunderstandings or weaknesses before they result in failure [Arnold and Pistilli(2012)], [Olivé et al.(2018)]. Other implementations aim to gain insights into student sentiments about specific parts of the course. This information allows educators to modify course structures, clarify assignment instructions, or offer additional support [Olivé et al.(2018)], [Ezen-Can et al.(2015)]. Recent implementations provide dashboards to visually present course data to educators [Krusche and Seitz(2018)], [Olivé et al.(2018)], [Dourado et al.(2021)], [Kaliisa and Dolonen(2023)].

The method we propose in this paper produces competence-related learning analytics data predominantly intended for educators' use. This data enables educators to identify the programming competences that students develop throughout the course and competence areas in which they encounter difficulties. Furthermore, as students receive feedback from the unit test results, the competence-based tests also inform the students about potential weaknesses and competences not yet developed. Currently, specialized dashboards for student use have not been developed, but their development is planned for the future.

### B. Competence-Based Learning

Competence-based learning is an educational approach that focuses on the application of the knowledge gained. It concentrates on the acquisition and application of skills, knowledge, and abilities necessary to successfully perform tasks

in specific areas [Rugelj et al.(2010)]. The approach differs from traditional educational approaches that primarily focus on the theoretical understanding of subjects. In competence-based learning, the focus is on students demonstrating their ability to apply what they have learned in real-world situations.

The importance of competence-based learning in higher education, particularly within the European Union, has been significantly influenced by the Bologna Declaration of 1999, which advocated a shift towards competence-based, student-centered educational models [Bergsmann et al.(2015)]. This paradigm shift has not only led to new pedagogical approaches but also fostered the development of strategies to assess the degree of achievement of competences developed by students [Alsinet et al.(2009)], [Anke and Schumann(2017)].

We recently developed a competence model for our introductory programming course. Figure 1 shows its metamodel based on the competence model presented by Bergsmann et al. [Bergsmann et al.(2015)]. Individual competences are grouped into competence areas. Competences can require or extend other competences. They consist of components, which are either skill or knowledge. Components have a competence level, representing the degree of expertise. Competence levels can be defined for both components, knowledge and skill, and may be specified numerically or verbally.

In our model, we defined competence areas for the different topics of our programming course, including conditions, loops, methods, and arrays. Individual competences are defined within the areas. In total, our model consists of 11 competence areas and 85 individual competences, corresponding to an average of 7.7 competences per area.

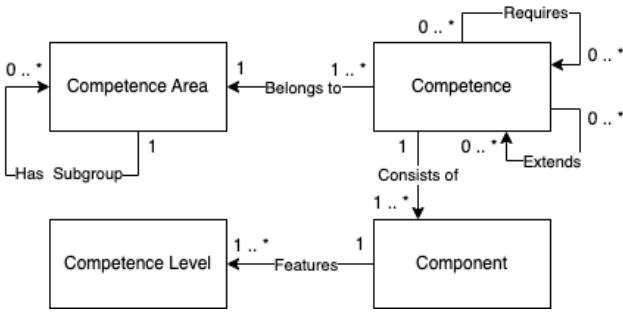


Fig. 1. Metamodel of our Competence Model based on [Bergsmann et al.(2015)].

#### IV. RELATED WORK

In our review of the literature, we identified different approaches that derive learning analytics data from unit-test frameworks such as JUnit.

Garmann [Garmann(2016)] introduces Graja, an automatic grading system based on unit tests. Garmann defines three distinct roles: exercise creators, responsible for designing assignments and developing corresponding tests; educators, who implement these assignments in their courses and have the option to manually modify grading; and students, completing these assignments. The system conducts an assessment of

submissions by comparing their output against a predefined sample solution and presents the differences to the educator for review.

In their work, Oster et al. [Oster et al.(2017)] introduce AuDoscore, a JUnit extension designed for the automated grading of Java assignments using unit tests. This system integrates with LMS to receive student submissions, which are then subjected to various tests. AuDoscore distinguishes two kinds of tests: public tests, where feedback is visible to students, and secret tests, with feedback accessible only to tutors. Educators are provided with a comprehensive table detailing the test results for all student submissions.

Gandraß et al. [Gandraß and Schmolitzky(2019)] developed an extension for Moodle that facilitates the execution of JUnit-based unit tests on student-submitted code. This tool provides students with instant feedback on their submissions. A notable aspect of Gandraß' system is its capability to function with incomplete Java classes, still offering valuable feedback. The extension's role within Moodle is primarily interactive, featuring a rich text editor for student use and displaying feedback. The grading process occurs externally: a server runs the predefined tests on the submissions and subsequently sends the results back to Moodle.

The majority of the learning analytics methodologies examined in our review focus on delivering analytics directly to students and supporting educators in grading. The literature primarily emphasizes evaluating student performance rather than exploring the root causes of student mistakes. Our search did not identify any literature addressing the assessment of competences through unit tests.

#### V. INITIAL ATTEMPTS FOR UNIT TEST-BASED LEARNING ANALYTICS

In this section we give an overview of our introductory programming course and describe how we tried, in a first attempt, to generate reasonable learning analytics data from unit tests integrated in our LMS ArTEMiS.

##### A. Course Overview

Our introductory Java programming course teaches students basic coding principles such as variables, conditions, loops and methods, including the basics of object-oriented programming. Students do not need any prior programming knowledge for our course as most students with prior programming experience from secondary education can skip the course. The course consists of two parts: The lecture, evaluated through an exam, and the exercise, evaluated through weekly assignments. To pass, students are required to achieve a minimum of fifty percent of the exam, and must also submit at least eight of the ten assignments ensuring that the cumulative points across these submissions reach or exceed fifty percent of the total points possible for these assignments.

ArTEMiS [Krusche and Seitz(2018)], the LMS we use throughout the course, supports both students and teachers with a variety of features, such as a web-based editor for students, automatic unit tests that are executed whenever a

student submits a (partial) solution, and dynamically generated statistics about the course for both individual students and entire course groups. Prior to the submission deadline, students are permitted to repeatedly submit their solutions to receive feedback from the automated unit tests. ArTEMiS also supports our tutors in evaluating the submitted assignments and assigning points to students based on their performance. Grading is therefore not automated, but heavily supported by the ArTEMiS system.

### B. Learning Analytics based on Unit Tests

As students heavily use ArTEMiS we had the idea to generate learning analytics data using the unit tests provided for the student assignments. These unit tests perform a variety of checks on the students submissions and provide them with feedback. Nevertheless, the unit tests aim at functional correctness and are not specifically designed in a competence model based manner.

To get a better understanding of the challenges our students face, we analysed the ArTEMiS unit test results for all submission in detail. To get a more comprehensive overview than that provided by individual unit tests, we defined error categories and manually mapped them to all unit tests. We categorized the errors into specific groups to facilitate the mapping of numerous individual errors into more tangible categories: (1) Functional correctness (2) Appropriate management of functional boundary conditions (3) Robustness with a focus on null pointer management and invalid parameter processing, and (4) Correct formatting of console output. A unit test can be assigned to multiple error categories.

In the following, we discuss the results from our analysis for the winter semesters in 2021 and 2022.

Figure 2 illustrates the frequency of submissions and corresponding failures for each error category over the ten assignments in the year 2021. As we typically experience a high dropout of students during the semester, we expect the rate of submissions and errors to increase at the beginning of the course, and fall off towards the end. This figure visualizes 10 programming assignments for a total of approximately 150 students.

The early exercises have a smaller amount of errors, which we assume to be because of simpler programming tasks in the beginning. Due to a smaller or sometimes non-existent test suite we cannot identify challenges students have to face in this phase of the course.

While we can identify a significant increase in both submissions and errors for assignments five through eight, we lack information to determine the cause for this increase. Our categorization of unit tests is too coarse-grained to help us here.

The decrease in submission for exercises 9 and 10 is explainable with our grading rules for the exercise - students are only required to submit eight out of the ten assignments to pass the class.

For the core exercises (exercises 4 to 8) We can see from the data that students make more mistakes and therefore submit

more solutions for grading, with the peaks of both submissions and errors correlating with each other. We can infer that harder tasks require more submissions until students achieve a correct implementation for the task. Since we can see a fairly even increase in errors and submissions, it stands to reason that students generally try to solve their assignment many times, and take many iterations to arrive at a satisfactory solution. Again, the categorization of unit tests does not help us to understand the real competence-related problems of the students.

We can gather from the data that functional correctness and robustness errors generally occur throughout the entire course, whereas formatting and border case errors showcase peaks in one specific assignment, exercise seven. This points us to a likely possibility that the assignment may contain new formatting requirements, which students did not need to implement before, or overly strict test cases in comparison to the rest of the course. While this information does not give us a clear indicator of problems students face in this assignment, we can use this information as a starting point for further manual investigation. Further review of this assignment shows that students often used array indices in the wrong order on the exercises on two-dimensional arrays in the assignment, which lead to an increase in *ArrayIndexOutOfBoundsExceptions* and incorrect formatting on outputs.

Figure 3 depicts the submission numbers and associated errors for the year 2022. There is an observable increase in submissions for the initial array-related assignment assignments handed out in the middle of the course, which is then followed by a gradual decrease. The final assignment has a modest increase in submissions, alongside a slight increase in errors.

In the year 2022, the submissions and errors again closely correlate with each other, with the middle assignments having both the highest submissions and errors. As in the previous year, the seventh exercise showcases an outlier in formatting errors. Since this mirrors our findings in the previous year's data and our test suite was updated for this year, we can rule out one of our assumptions, that the test cases were too strict,

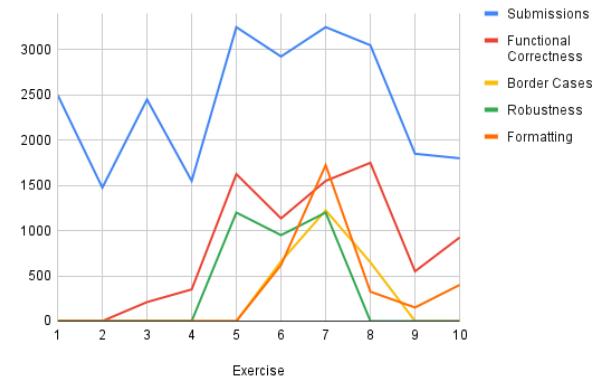


Fig. 2. Unit Test-Based Analysis of Winter Semester 2021

and with a high degree of confidence gather that students struggled with the formatting requirements with the new topic of matrices. As such, we should review the assignment and offer students additional support with matrix outputs to the command line. As border cases also show a similar increase for the seventh assignment in both years, we can also derive that students still need to practice properly handling illegal inputs for matrix operations.

In summary, the analysis of error categories only offers limited capabilities for extracting meaningful conclusions regarding the challenges students have to struggle with. Furthermore, there is a limitation in the utility of the broad error categories we used, as they fail to capture the different programming challenges students face. Difficulties in implementing functionalities are uniformly categorized under functional correctness, regardless of the diverse implementation demands of the exercises.

The pre-existing unit tests were not originally designed for learning analytics purposes. Consequently, tests may belong to several categories, leading to the same failures being counted in multiple categories. For instance, unit tests that assess both functional correctness and proper formatting make it impossible to sometimes cover both functional correctness and correct formatting, making it impossible to differentiate between the two.

### C. Conclusions from the Initial Analysis

Our analysis indicated common difficulties faced by students, highlighted by the increase in errors on assignments dealing with two-dimensional arrays. However, it proved challenging to isolate specific trouble areas, as unit tests largely identified errors in console output formatting and general issues of robustness and functional correctness. This data is useful for identifying assignments and tasks that may require additional instructional focus, yet there is a need to refine our analytical methods to extract more targeted learning analytics data.

During the preliminary analysis, error categories were retrospectively assigned to unit tests by hand. The main intent of

the unit tests was to support the grading process for tutors by checking for frequent errors and functional correctness, rather than offering instructive feedback to students. This ensured comprehensive coverage by our unit tests but also led to redundancies, with numerous tests assessing the same elements such as null-checks and edge case management, resulting in their classification into identical error categories.

From the analysis of this data, we conclude that enhancing our understanding of the students' learning progress requires the alignment of unit test labels with the specific competencies described in our curriculum framework, rather than with broad error categories. By correlating unit tests to particular competencies related to the central topics of the course, we hope to be able to identify the specific areas within each topic where students struggle.

In the following section, we will introduce a methodology for developing competence-oriented unit tests for learning analytics. Applying this approach, we will re-examine the student outcomes for select assignments to determine if a shift towards competence-based unit tests provides more insightful learning analytics data.

## VI. METHOD FOR COMPETENCE-BASED LEARNING ANALYTICS

Based on our experience and based on the results of the initial analysis, we propose the **Competency Based Learning Analytics** method (CBLA-method) for systematically deriving programming assignments based on competences. (see Figure 4). The emphasis of this method is to derive assignment and programming task specifications that match required competences and to implement unit tests in a way that competence related learning analytics data can be systematically collected. These unit tests are of course also used for autograding of programming tasks. The autograding aspect is out of scope for this paper. The method is explained in more detail in the following subsections.

### A. Initial Exercise Creation

Each assignment consists of one or more programming tasks. Programming tasks help students understand, learn and practice one or more competences of the competence model. To allow the best possible competence support, when creating assignments, educators should initially decide which primary competences the students should be confronted with in the assignment as well as the degree of intensity. After the set of primary competences of an exercise has been chosen, educators develop a tangible problem description that students have to solve in the assignment. In this process, educators have to assign competences to specific programming tasks. This approach makes it easier to more specifically test later, whether a specific competence, e.g. *array declaration*, is achieved by a student or not. At this stage of exercise development, it helps to design the exercises and programming tasks in a way that all needed primary competences can be trained and tested. Other educators should also review the programming tasks with the assigned competences in mind. Therefore, the

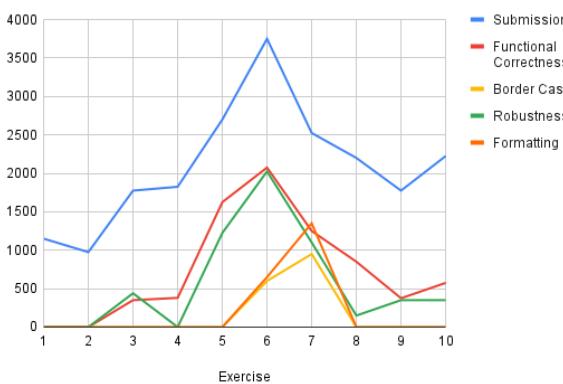


Fig. 3. Unit Test-Based Analysis of Winter Semester 2022

programming task descriptions have to be developed, and the primary competences are assigned to these programming tasks (see ① in Figure 4). Educators should also try to develop sample solutions for these tasks at this stage to cross-check, whether the assigned primary competences fit, tasks are not too complex, etc. (see ② in Figure 4). This iterative process is repeated until all primary competences of the assignment are covered in a balanced way by the individual programming tasks.

We then annotate the competences in the exercise specification, marking specific sections in the text referring to the integrated competences. It should be noted that we opted to only annotate the primary competences relevant for the assignment at this stage of the method, secondary competences are added later.

### B. Solution Creation and Finetuning

Having finalized the assignment specification and therefore the programming task specifications, educators have to elaborate and enhance the initial sample solution and if necessary the programming task descriptions.

As a starting point for the refinement, the educators should assign secondary competences to the initially developed sample solutions for the programming tasks (see ③ and Figure 4). Secondary competences are not "second class" competences, but competences students should already have learned and practised. These secondary competences were primary competences in earlier assignments. This typically helps with

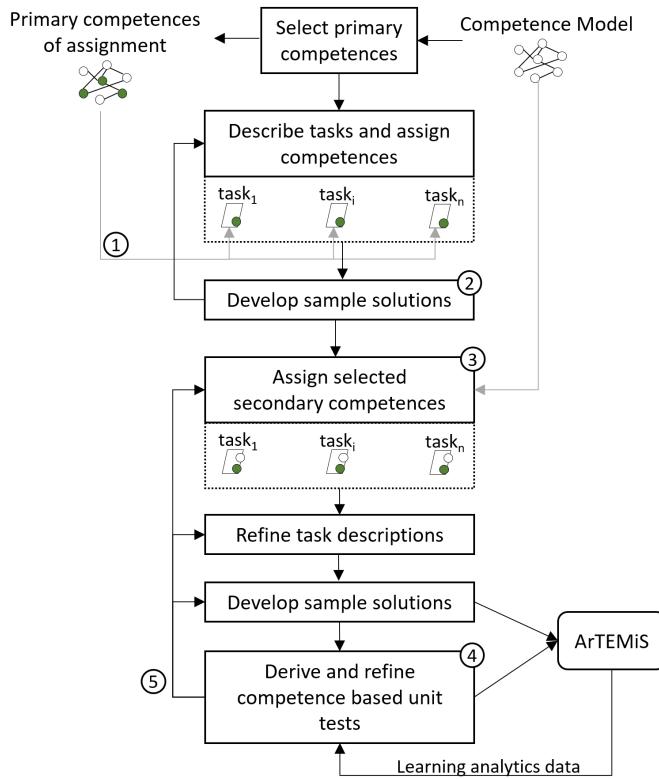


Fig. 4. CBLA Method Overview

managing the complexity of the programming tasks. Additionally, from a competence perspective, educators get a good understanding of the secondary competencies involved in a programming task.

In the process of enhancing and providing a sample solution for a programming task, educators may discover further issues in the exercise specification, which should be fixed. However, this does not mean that educators should adjust the programming task specification to match the created sample solution; if issues are found that require changes, the sample solution should be redone or enhanced.

In this iterative refinement process the secondary competences needed for an exercise, i.e., for the programming tasks, have to be adopted and adjusted based on the developed sample solution.

In this iterative refinement process, other educators, ideally different from those who reviewed the exercise specification, should review the enhanced exercise specification, assigned secondary competences as well as the sample solutions. The specifications and solutions should be enhanced based on these reviews until both the author and the reviewers are satisfied with the result. In this context, educators should aim producing a solution that students could also create on their own with the knowledge and understanding they should have based on their learning progress.

In this iterative process, we annotate the implemented primary and secondary competences in the sample solution code, for example by writing comments for specific code snippets that implement each individual competence. This will help us later in the systematic derivation of competence based unit tests.

### C. Unit Tests Creation

Having now achieved well reviewed assignment and programming task specifications, annotated with primary and secondary competences, educators can now start to design and implement competence-based unit tests (see ④ in Figure 4).

We use the defined primary and secondary competences from the specification to design the respective unit tests. Educators should do this in the following order:

- 1) Each primary competence must have at least one unit test. For more complex competences (e.g. proper iteration of two-dimensional arrays) more unit tests will be necessary.
- 2) Each secondary competence can have a least one unit test. Typically, this is necessary anyway to test for the functional correctness of the solution, but has the advantage that we can collect learning analytics data on the mastery of the competences previously acquired by the students. While testing of primary competences is mandatory, testing of secondary competence can be decided individually.
- 3) Check, from a black box perspective, whether any additional test cases are necessary to check the correctness of the solution of the students. Educators should add additional tests and try to assign them to one of the

primary or secondary competences. If competences are missing, they should be systematically added to the specification of the assignment or programming task (see ⑤ in Figure 4).

- 4) Perform a thorough coverage analysis of the sample solution to identify untested code segments. This helps educators identifying dead code or missing test cases. Based on the analysis of the coverage data, the sample code can either be enhanced (in case of dead code) or additional unit tests have to be defined. Ideally, as described in the previous step, with necessary adaptions of task descriptions and selected secondary competences.
- 5) In a last step, the educator should analyse, whether there are unit tests that test more than one competence. If this is the case, educators should try to split unit tests in a way that a unit test only covers one competence. Nevertheless, this is not mandatory - splitting only makes sense in cases where this does not lead to massive code duplication in the unit tests.

To facilitate the implementation of robust unit tests, we introduced a small framework to ensure that students always get reasonable feedback. Typical examples that have to be considered are: (1) proper feedback in cases where we cannot even compile our test suite together with the student code because of syntax errors in the student submission. (2) In many programming tasks, we ask students to implement a Java method with a given method signature. We use Java reflection mechanisms to find out whether the method provided by the student conforms to the needed signature and give specific hints if it does not.

#### D. An Example Implementation

As we have the ArTEMiS data of the student assignments (as described in Section V) we wanted to apply the CBLA-method for two programming exercises dealing with one dimensional and two-dimensional arrays. These two assignments (assignment 6 and assignment 7) consist in total of seven programming tasks. For these programming tasks we identified the following primary competences of our competence model (see Section III-B):

- *Array declaration (a-decl)*: Students can declare and create one-dimensional arrays.
- *Array initialization (a-init)*: Students can initialise one-dimensional arrays. They are also able to assign values to array cells.
- *Process all array elements (a-iter)*: Students can iterate through arrays using loop constructs.
- *Passing arrays as parameter (a-para)*: Students can use arrays as parameters in methods.
- *Arrays and null values (a-null)*: Students understand the value ‘null’ in the context of arrays.
- *Array specific runtime errors (a-runt)*: Students understand the common runtime errors in the context of arrays and are able to fix the underlying causes.
- *Two dimensional arrays (a-matr)*: Students understand multi-dimensional arrays and can use them in code.

As we worked with existing assignments and programming tasks, we could of course not alter the initial programming task description. Therefore, the first steps of the method that comprise the selection of the primary competences as well as the formulation of the initial programming tasks and sample solutions (guided by our method) could not be done. Nevertheless, analysis shows us (see Figure 5) that we covered all primary competences in the competence group *arrays* in our assignment specification, though for some competences, e.g. a-runt and a-matr, we would put more emphasis in the programming tasks in the future. We also assigned the major secondary competences to the programming tasks. In retrospect, this shows that e.g. the programming task 7.4 with 5 primary competences and 13 secondary competences is probably too complex and should be reduced in the future. Figure 5 shows not only the competences but also the number of implemented unit tests. For the unit tests we distinguish between unit tests focusing on checking primary competencies, unit tests focusing on secondary unit tests and finally other unit tests that deal with the general correctness of the solution.

The data evolved over time, as we applied the CBLA-method and the data shown in Figure 5 is the data we obtained at the end of the experiment, i.e., after having made some adoptions to our tests. Here are some remarks on this data:

The CBLA-method guided us to achieve full coverage of our sample implementations. For the programming task 7.1 we could not achieve full coverage for the time being. The reason for this is that although the implementation is correct, the sample solution implements a defensive coding practice with respect to parameter checking that was not required in the programming task description. As we did not check this (it was not required from the specification) we could not achieve 100% coverage. As a consequence, we should either adopt the task description or our sample implementation.

The coverage data helped us to clearly identify a missing unit test to cover a specific case in the programming practice 6.3. We corrected this and now achieve 100% coverage.

For some competences (e.g. a-decl and a-init) we had one unit test that checked both aspects. This makes it difficult to decide which of these primary competences are not fulfilled in case this unit test fails. We therefore split these unit tests to be able to distinguish these two principles more clearly.

We followed the practice of assigning secondary compe-

	<b>6.1</b>	<b>6.2</b>	<b>6.3</b>	<b>6.4</b>	<b>7.1</b>	<b>7.3</b>	<b>7.4</b>
<b>Competences</b>							
# Primary	5	4	4	3	6	6	5
# Secondary	6	12	9	12	7	6	13
<b>Unit Tests</b>							
# Primary	7	4	6	2	8	4	5
# Secondary	4	0	0	0	0	0	0
# Others	5	16	7	9	14	16	2
<b>Coverage</b>							
Line%	100%	100%	100%	100%	96%	100%	100%

Fig. 5. Competences and Unit Test for Assignments 6 and 7

tences also to unit tests that test for a primary competence. Our analysis shows that this has little value, as there is typically no direct relation between the unit test, but rather an indirect one. If a student e.g. is capable of iterating over a one dimensional array, they obviously have the competency to deal with loops. Though, in cases where a student fails in iterating the array, they still could be capable of dealing with loops. It is therefore a decision whether learning analytics data should be provided for secondary competences or not. If educators want to concentrate on these learning analytics data for secondary competences, the same rules should apply as for primary competences. We did this in our example for the secondary competencies in the programming task 6.1. and derived 4 unit tests that test for secondary competencies.

Since the assignments in our case study were not created with testable competences in mind, we often only have one method to call in our unit tests. This leads to some programming tasks having a lot of unit tests for one competence, to ensure properly detecting if a submission achieved the competence. This will skew the visualisation if we only consider the total amount of failed unit tests. As such, we need to normalize our failed total not only to the total users, but also to the amount of unit tests for the specific competence.

Each developed unit test is annotated to describe what it does and which competences are tested. This not only serves the purpose of documentation, but will help us in the future to enhance the CBLA-method.

## VII. REEVALUATING ARRAY ASSIGNMENTS

In order to test the application of the CBLA-method on the learning insights from generated learning analytics data, we chose to focus on the two array assignments (assignment 6 and assignment 7), as we could identify a spike in both submissions and errors for these assignments in the previous analysis.

To allow proper comparison between the two years, we only considered programming tasks that were available in both years, namely 6-1, 6-2, 6-3, 6-4, 7-1, 7-3 and 7-4. We skipped the programming task 7-2 as this task was different in the two years and therefore not comparable. Apart from 7-1 and 7-4, which focus on two-dimensional arrays, the other exercises are about one-dimensional arrays.

The data has been normalised per student. As such, a value of 0.1 in a competence means that for that specific exercise, 10% of students have failed the competence. While we do employ multiple tests per competence due to the restrictive exercise design, we have also considered this in our analysis by weighing tests in respect to the total amount of tests per competence.

By reviewing the competences in the winter term of 21/22 (see Figure 6, where the x-Axis denotes the individual programming tasks and the y-axis denotes the ratio of submissions with errors to the total number of submissions), we can immediately tell that the competences *a-decl* and *a-init* are not often tested for, and even when they are, are usually not failed. This is because many exercises provide the array that

needs to be modified already and require the reference to be changed, and thus do not require students to declare their own array. Assigning arrays is also often done correctly, as most errors that could be made, such as trying to assign values to a cell in a null array or to a cell with an invalid index, are already covered by competences *a-null* and *a-runt*.

The competence *a-null* is steadily decreasing, as students submit exercises. As the competence *a-null* is the competence regarding proper null handling, we can derive from this, that students learn to check for null in the first assignment, where they may still make errors, and then learn to consider the possibility of null as a parameter in the second assignment.

The competence *a-runt*, which checks if the students properly address potential array related runtime exceptions, can not be explicitly tested in the first set of assignments, due to the programming task specification. However, in the exercises regarding two-dimensional arrays, this is failed as exercises gain difficulty, and as students may be confused and mix up the first and second index when trying to access values, which could cause an exception.

The competence *a-iter*, looping through arrays, is also decreasing as students solve exercise tasks, but spike again in both the two-dimensional array exercise with up to 23% in 7-1, and increase in the more difficult one-dimensional array exercises, 6-4 and 7-3. These exercises require a bit more in addition to just iterating through the array, such as inverting its order or comparing values.

The competence *a-para* is implicitly required for other competences to be evaluated, and thus is hard to test for in isolation with the given programming task descriptions. However, we assume that the increased amount of runtime errors and non-compiling submissions in later exercises stem from students not handling the parameterised arrays properly.

The competence *a-matr* is a broad competence referring to students being able to properly handle multi-dimensional arrays in declaration, initialisation, assignment and mutation. As such, this can only be tested for the exercises 7-1 and 7-4, the only exercises involving two dimensional arrays. The exercise design of 7-4 leaves little room for explicit testing that is not already covered by other competences, so it is omitted

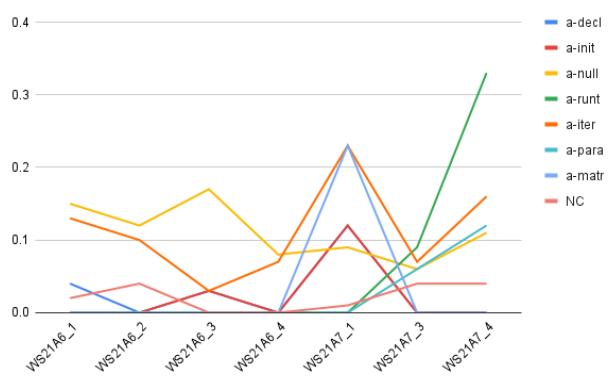


Fig. 6. Primary Competences in Study Year 21/22

for this exercise. We can identify a large spike for 7-1 at 23%, which is not surprising as it is many students' first practical confrontation with multi-dimensional arrays.

In the winter term 22/23 (see Figure 7), we see a similar progression for the primary competences *a-decl* and *a-init*, for the same reasons outlined above.

The competence *a-null* remains constant at around 10% in the first assignment, then peaks in the first multi-dimensional array exercise at 15% before decreasing over the course of the second assignment. As opposed to the previous year, students seem to get a good grasp on the concept of null robustness for one-dimensional arrays, but struggle a bit more with null in multi-dimensional arrays. After the first multi-dimensional array exercise, 7-1, students manage to hit an all-time low in failure rate in the second multi-dimensional array exercise, 7-4, with 4%.

Similarly to the previous year, the competence *a-runt*, which checks if the students properly address potential array related runtime exceptions, also increased over the course of the second assignment, maxing out at 35% in 7-4. As the exercises did not change, we can assume that students similarly got confused about the indices in the more difficult exercises.

The competence *a-iter* remains consistent around 8% throughout assignment one, then peaks in the first multi-dimensional array exercise at 28% before decreasing over the course of the second assignment, mirroring the learning progression we observed for the competence *a-null*. Compared to the previous year, neither *a-null* nor *a-iter* increase with the second multi-dimensional array exercise, 7-4.

The competence *a-par*a rarely fails in the last two exercises, and when it does, it is likely due to students handling parameterised one-dimensional arrays as multi-dimensional arrays and vice-versa.

The competence *a-matr*, just as in the previous year, peaks in the first multi-dimensional array exercise with 27%.

### VIII. CONCLUSION

From the practice of teaching introductory programming courses, we recognized in the past that students fail to finish

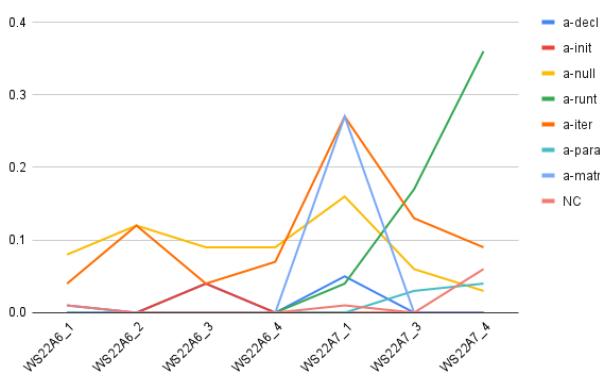


Fig. 7. Primary Competences in Study Year 22/23

the course. Besides more social and personal oriented reasons (e.g., underestimation of the effort, bad personal time management) the difficulty of the subject itself is a reason for students to fail. The paper contributes the CBLA-method for developing programming assignments as well as individual programming tasks based on competences students have to acquire during the course. The method on the one hand helps with identifying and selecting necessary competences for an assignment, which helps educators to balance content, amount and complexity of entire assignments and individual programming tasks. Furthermore, the CBLA-method guides educators in systematically deriving unit tests in a way that learning analytics data can be generated that helps educators in judging which competences are difficult to learn for students.

Regarding **RQ1**, our findings suggest that conventional unit tests, which do not explicitly adopt a competence-oriented perspective on student submissions, provide limited competence-related insights. The data derived from these tests primarily offer insights into students' abilities to develop correct software in principle and their readiness to address edge cases. Additionally, the submission and error data gives hints about programming tasks that were difficult for students and those they could deal with more easily. More detailed insights cannot be found.

Regarding **RQ2**, we developed the CBLA-method and partly applied this method on two assignments consisting of seven programming tasks. Owing to the utilization of pre-existing assignments and programming tasks, the initial phases of the method, specifically those concerning the selection of primary competences and the systematic derivation of programming tasks, were not executed. Our analysis focused on evaluating the specified programming tasks. Findings from the application of our method indicated that certain tasks were overly complex, requiring an extensive set of primary and secondary competences for resolution. Additionally, it was observed that some primary competences were underrepresented, resulting in insufficient learning opportunities within the existing programming tasks. This analysis suggests that these limitations might have been mitigated through the full application of the CBLA-method.

The second part of the CBLA-method deals with the development of unit tests that help with collecting competence related learning analytics data. We systematically applied the method and could considerably enhance the existing unit tests in different ways for our case study: (1) better coverage and consideration of a corner case for a programming task, (2) simplification of the sample solution, and (3) smaller unit tests that focus on individual competences. Furthermore, applying the method for developing the unit tests also revealed subtle deficiencies in the programming task specifications that made it difficult for some competences to be explicitly tested.

Regarding **RQ3** we can conclude from the analysis (see Figures 6 and 7) that the learning analytics data provides far more insights concerning the learning problems of students from a competence perspective compared to the initial data provided. For selected primary competences, e.g. dealing with

null values in the context of arrays, we can see a clear positive learning trend. For other competences (e.g. iterating over arrays) we can deduce that the mastery depends also on other factors, i.e. other secondary competences that are needed for a programming task. This outcome is encouraging, as we did not yet fully apply the CBLA-method, e.g., we did not cover to a large extent the secondary competences by explicit unit tests. This would be of particular interest, as it would show whether these secondary competences (that were learned earlier in the course) are already mastered by the students or whether their insufficient mastery is the main reason for not being able to learn the new primary competences.

## IX. FUTURE WORK

We plan and already started to fully apply the CBLA-method for our introductory Python programming course in the next summer semester for business administration students, which is a mandatory course for them. Besides helping us to enhance the CBLA-method this broad application will help us to judge the overall suitability of the method in a practical setting, as a couple of educators will apply this method. We will explicitly monitor the effort needed to apply the CBLA-method. Applying the method should provide us with valid and meaningful learning analytics data for approximately 150 students. Furthermore, working with the gained data will help us with deciding whether focusing on competence based learning analytics data is an efficient and effective way for gaining a deeper understanding of the students' learning problems.

In this paper and as you can see from the visualization of our data (e.g. Figure 6) it is evident that our current visual representations are rudimentary and require refinement. We will therefore experiment with different types of visualizations that help us to gain better learning analytics insights. Additionally, we aim to innovate by integrating core learning analytics data with student-specific information, such as gender, prior programming experience, and the type of secondary education received. This integration seeks to uncover correlations between these variables and the competences acquired in our introductory course. Our ultimate goal is to develop a comprehensive learning analytics dashboard that provides valuable insights for us and other stakeholders, such as students and program directors, facilitating data-driven decision-making in learning.

## REFERENCES

- [Alsinet et al.(2009)] Teresa Alsinet, David Barroso, Ramón Béjar, and Jordi Planes. 2009. A Formal Model of Competence-Based Assessment. IOS Press, NLD, 428–436.
- [Anke and Schumann(2017)] J.M. Anke and M. Schumann. 2017. Examining An Extended Technology Acceptance Model For Competence-Based Assessment. In *ICERI2017 Proceedings* (Seville, Spain) (*10th annual International Conference of Education, Research and Innovation*). IATED, 658–667. <https://doi.org/10.21125/iceri.2017.0258>
- [Arnold and Pistilli(2012)] Kimberly E. Arnold and Matthew D. Pistilli. 2012. Course Signals at Purdue: Using Learning Analytics to Increase Student Success. In *Proceedings of the 2nd International Conference on Learning Analytics and Knowledge* (Vancouver, British Columbia, Canada) (*LAK '12*). Association for Computing Machinery, New York, NY, USA, 267–270. <https://doi.org/10.1145/2330601.2330666>
- [Bennedsen and Caspersen(2019)] Jens Bennedsen and Michael E. Caspersen. 2019. Failure rates in introductory programming: 12 years later. *ACM Inroads* 10, 2 (April 2019), 30–36. <https://doi.org/10.1145/3324888>
- [Bergsmann et al.(2015)] Evelyn Bergsmann, Marie-Therese Schultes, Petra Winter, Barbara Schober, and Christiane Spiel. 2015. Evaluation of competence-based teaching in higher education: From theory to practice. *Evaluation and Program Planning* 52 (2015), 1–9. <https://doi.org/10.1016/j.evalprogplan.2015.03.001>
- [Dourado et al.(2021)] Raphael A. Dourado, Rodrigo Lins Rodrigues, Nivan Ferreira, Rafael Ferreira Mello, Alex Sandro Gomes, and Katrien Verbert. 2021. A Teacher-Facing Learning Analytics Dashboard for Process-Oriented Feedback in Online Learning. In *LAK21: 11th International Learning Analytics and Knowledge Conference* (Irvine, CA, USA) (*LAK21*). Association for Computing Machinery, New York, NY, USA, 482–489. <https://doi.org/10.1145/3448139.3448187>
- [Ezen-Can et al.(2015)] Aysu Ezen-Can, Kristy Elizabeth Boyer, Shaun Kellogg, and Sherry Booth. 2015. Unsupervised modeling for understanding MOOC discussion forums: a learning analytics approach. In *Proceedings of the Fifth International Conference on Learning Analytics And Knowledge*. ACM, Poughkeepsie New York, 146–150. <https://doi.org/10.1145/2723576.2723589>
- [Gandrab and Schmolitzky(2019)] Niels Gandrab and Axel Schmolitzky. 2019. Automatisierte Bewertung von Java-Programmieraufgaben im Rahmen einer Moodle E-Learning Plattform. Gesellschaft für Informatik e.V. <https://doi.org/10.18420/ABP2019-1>
- [Garmann(2016)] Robert Garmann. 2016. Graja - Autobewerter für Java-Programme. <https://www.semanticscholar.org/paper/Graja-Autobewerter-f>
- [Joksimovic et al.(2019)] Srecko Joksimovic, Vitomir Kovanovic, and Shane Dawson. 2019. The Journey of Learning Analytics. 6 (01 2019), 37–63.
- [Kaliisa and Dolonen(2023)] Rogers Kaliisa and Jan Arild Dolonen. 2023. CADA: a teacher-facing learning analytics dashboard to foster teachers' awareness of students' participation and discourse patterns in online discussions. *Technology, Knowledge and Learning* 28, 3 (Sept. 2023), 937–958. <https://doi.org/10.1007/s10758-022-09598-7>
- [Krusche and Seitz(2018)] Stephan Krusche and Andreas Seitz. 2018. ARTEMiS: An Automatic Assessment Management System for Interactive Learning. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education* (Baltimore, Maryland, USA) (*SIGCSE '18*). Association for Computing Machinery, New York, NY, USA, 284–289. <https://doi.org/10.1145/3159450.3159602>
- [Nikula et al.(2011)] Uolevi Nikula, Orlena Gotel, and Jussi Kasurinen. 2011. A Motivation Guided Holistic Rehabilitation of the First Programming Course. *ACM Transactions on Computing Education* 11, 4 (Nov. 2011), 24:1–24:38. <https://doi.org/10.1145/2048931.2048935>
- [Olivé et al.(2018)] David Monllaó Olivé, Du Q. Huynh, Mark Reynolds, Martin Dougiamas, and Damyon Wiese. 2018. A Supervised Learning Framework for Learning Management Systems. In *Proceedings of the First International Conference on Data Science, E-Learning and Information Systems* (Madrid, Spain) (*DATA '18*). Association for Computing Machinery, New York, NY, USA, Article 18, 8 pages. <https://doi.org/10.1145/3279996.3280014>
- [Oster et al.(2017)] Norbert Oster, Marius Kamp, and M. Philippse. 2017. AuDoscore: Automatic Grading of Java or Scala Homework. *Revista Brasileira de Psiquiatria* (2017). <https://www.semanticscholar.org/paper/AuDoscore>
- [Rugelj et al.(2010)] Jozé Rugelj, Sven Knockaert, Roel Van Steenberghe, Luk Schoofs, Janne Salonen, Kari Björn, Jose L Marzo, and Carlos Vaz de Carvalho. 2010. Competence based joint study program on advanced networking technologies for blended learning. In *2010 9th International Conference on Information Technology Based Higher Education and Training (ITHET)*. 372–375. <https://doi.org/10.1109/ITHET.2010.5480100>

# Acknowledging Good Java Code with Code Perfumes

Philipp Straubinger  
University of Passau  
Passau, Germany

Florian Obermüller  
University of Passau  
Passau, Germany

Gordon Fraser  
University of Passau  
Passau, Germany

**Abstract**—Java remains one of the most popular programming languages in education. Although Java programming education is well supported by study materials, learners also need more immediate support on the problems they face in their own code. When this support cannot be offered by educators personally, learners can resort to automated program analysis tools such as linters, which provide feedback on potential bugs or code issues. This is constructive feedback, but it may nevertheless feel like criticism. This paper introduces *code perfumes* for Java, a simple program analysis technique similar to linting, but commanding the correct application of good programming practices. We present a catalogue of 20 Java code perfumes related to common Java language constructs for beginner to immediate learners. Our evaluation shows that these code perfumes occur frequently in learners' code, and programs with more code perfume instances tend to have better functionality and readability. Moreover, students who incorporate more code perfumes tend to achieve higher grades. Thus, code perfumes serve as a valuable tool to acknowledge learners' successes, and as a means to inform instructors about their learners' progress.

**Index Terms**—Linting, Code Quality, Java, Education

## I. INTRODUCTION

Java is one of the most frequently used languages to teach programming [1], [2], and teaching materials and self-study materials are abundantly available. However, independently of the maturity of these materials, students may struggle in their attempts to produce functioning programs. Even when they produce code that implements the desired functionalities, this is often prone to quality problems, which the students again may struggle to identify and fix on their own [3]. In a self-study scenario, or when tutors are overwhelmed with large classrooms and cannot provide help and feedback promptly, learners can resort to automated program analysis tools such as *SpotBugs*<sup>1</sup> or *SonarLint*<sup>2</sup>. Feedback provided by these tools on quality issues can help learners produce better code and supports their learning processes, and tutors benefit from insights into their students' current skills and problems or misunderstandings. However, such automated feedback tools mostly focus on pointing out negative aspects.

While this kind of corrective feedback was shown to be very helpful in gaining further cognitive skills [4], purely negative feedback may be detrimental to self-efficacy and decrease intrinsic motivation [4]. In contrast, positive feedback

is considered to have better effects on motivational aspects [5]. Consequently, for effective learning feedback should consist of information about errors as well as correct behaviour to address both cognitive and motivational aspects. To this end the concept of *code perfumes* was recently introduced for the block-based Scratch programming language [6] as a means for automated positive feedback: Code perfumes are patterns of code that are considered as good programming practices or provide evidence of correctly applied code constructs, implying an understanding of a certain programming concept.

In this paper, we introduce the concept of code perfumes to the domain of text-based programming languages by proposing a set of concrete code perfumes for Java. For example, the following code shows a student approach to programming a scoring method in a chess game assignment. The first line in the method retrieves the colour of the human player from the board. However, if the `ChessBoard` object is `null` this method call results in a `NullPointerException`.

```
public static double score(ChessBoard board) {  
    Color hColor = board.getHumanColor();  
    // Method logic...  
}
```

A better solution is to use *defensive programming* by checking parameter objects for `null` values before working with them. The *defensive null check* code perfume acknowledges if learners correctly apply this approach by checking whether the `ChessBoard` object is `null` at the beginning of the method, and if so reacting with an `IllegalArgumentException`:

```
public static double score(ChessBoard board) {  
    if (board == null) {  
        throw new IllegalArgumentException(  
            "Board is null!");  
    }  
    Color hColor = board.getHumanColor();  
    // Method logic...  
}
```

We propose a total of 20 code perfumes covering various coding concepts, and evaluate them on a dataset of 816 student submissions to programming assignments in a university course. In detail, the contributions of this paper are as follows:

- We introduce the concept of code perfumes as a means of positive feedback to Java.
- We describe and implement a catalogue of 20 code perfumes from five categories.

<sup>1</sup><https://spotbugs.github.io/>, last accessed 08.06.24

<sup>2</sup> <https://www.sonarsource.com/products/sonarlint/>, last accessed 08.06.24

- We empirically evaluate how frequently these code perfumes occur in student code.
- We empirically evaluate how the occurrence of code perfumes relates to correctness of the programs.

Our investigation shows that most of the proposed code perfumes occur frequently in Java programming assignments. Furthermore, our results suggest a relation between the occurrence of code perfumes and correctness in the form of functionality, readability and grades. These encouraging findings suggest that code perfumes can provide positive feedback for Java and inform teachers about the learning progress of students.

## II. BACKGROUND

### A. Code Quality Problems

Even though learning to program may be hard, most students will manage to produce functioning code after some time. However, research has shown that even functionally correct programs show quality problems and that students need feedback to stop producing more [3]. For example, a common type of quality problem is *code smells*, which are code idioms that increase the likelihood of errors in future program edits or decrease the readability and understandability of the code [7]. Prior research has shown that code smells have negative effects on learning to program as they meddle with the learners' ability to modify given code [8], [9]. *Bug patterns*, on the other hand, are aspects of code that are likely to lead to undesired behaviour (i.e., bugs or defects) [10]. In particular, misconceptions of learners often manifest in bugs that follow recurring bug patterns [11], [12], [13].

While finding and removing such quality problems is inherent to coding and learning to program, tools can offer support to learners as well as teachers. Static analysis tools (i.e., linters) like *SpotBugs* and *SonarLint* can automatically detect these quality problems and generate feedback, which has been shown to help students' code quality skills [14]. Furthermore, static analysis of code patterns is part of automated feedback and grading tools like *ArTEMiS* [15] providing information to teachers for assessing their students.

### B. Feedback

Feedback generated by automated program analysis tools is returned regardless of other characteristics of the student and is accordingly perceived as less threatening [5]. No hand raising is needed, thus preventing peer stigmatisation [16]. Computer generated feedback mainly focuses on bad coding practices or missing functionalities [1]. Such negative feedback may be useful in programming education in multiple ways: Teachers assessing learners' code and supporting students with their individual problems when programming need to know about their students' misunderstandings and currently lacking skills and tools that can help in analysing those [17], [18], [19]. Novices can benefit as corrective feedback helps to acquire further knowledge and cognitive skills [4], [14].

On the other hand, positive feedback is important as it leads to higher intrinsic motivation than negative feedback [5]. Previous research also suggests that more motivated learners

process all kinds of feedback better [20]. This in turn would possibly lead to even higher effects of the corrective feedback. Unfortunately, positive feedback generated by automated tools is rare [1]. Most attempts to introduce positive aspects to automated feedback are task-specific and give information about what parts of a task are correct [21]. While this is useful guidance on how to solve specified tasks, it gives no information about general aspects, like style or quality. One attempt to address this deficiency in the context of Java is the use of gamification for reducing warnings produced by analysis tools [22]. While this keeps motivation high [23] and leads to better code, it only minimises the negative feedback and does not contain any information about good coding parts.

Positive feedback is more common in educational block-based programming. For the block-based programming language Scratch [24] the Dr. Scratch [25] tool displays praise in the form of a short text such as "You're doing a great job. Keep it up!!!" and returns points on computational thinking concepts<sup>3</sup>. The LitterBox [26] static analysis tool can not only point out bug patterns and code smells but also detects *code perfumes*, which are code idioms indicating good programming practices or code showing understanding of certain programming concepts [6]. Code perfumes can be seen as the opposite of code smells or bug patterns and are not bound to a specific task but can be applied to any given project. They have been shown to appear frequently in Scratch projects and can be used as an indicator of correctness, as projects having a higher count of code perfumes tend to be more correct [6].

While Java is still one of the most used languages to learn programming [1], [2], similar positive feedback is absent, which may limit the learning effects of automated tools like ArTEMiS [15]. To address this problem, we introduce code perfumes to this domain such that holistic feedback combining positive and negative aspects can be generated automatically.

## III. CODE PERFUMES FOR JAVA

Code perfumes are a general concept applicable to any programming language or skill level. In this paper, we target Java learners who already have some basic understanding, and are able to produce code using external APIs or design patterns. A specific concern related to such learners is that even though they are familiar with the syntax, they struggle to assess the quality of their code [27]. However, the code perfumes we define may also appeal to more experienced programmers.

Code perfumes are expected to have mostly positive effects on the code quality, be discernible in source code without execution, and align with language-dependent concepts. However, programmers must carefully assess their code and determine whether applying a code perfume is appropriate in a specific context, or if other solutions should be considered. This problem could be exacerbated by false positives, which is why implementations of checks for code perfumes need to aim to minimise false positives while acknowledging the need for careful application.

<sup>3</sup><http://www.drscratch.org/>, last accessed 08.06.24

We categorise code perfumes based on their purpose:

- **Solution Patterns:** These are code structures that solve undesirable code patterns (e.g., code smells or bug patterns). They directly address and avoid matching negative patterns, but not all bug patterns imply solution patterns [28]. Ideal solution patterns provide additional positive benefits beyond eliminating negative patterns.
- **Best Practices:** Established best practices in Java development can serve as the foundation for code perfumes. Capturing best practices in code patterns allow learners to leverage the results of experienced programmers [29].
- **Contracts or Conventions:** Java contracts, often documented in method descriptions, form another basis for code perfumes. Adhering to contracts, especially from the standard library is crucial for developing correct code and effective collaboration in development teams [30].

To define code perfumes for Java, we draw inspiration from the SonarRules collection of negative patterns by SonarSource[45] and the popular book “Effective Java” by Joshua Bloch [31]. SonarRules provides established code smells, bug patterns, and vulnerabilities, influencing the creation of “solution pattern” code perfumes. “Effective Java” offers best practices and guidance for Java programming, focusing on topics such as design patterns, concurrency, and Java standard library traits. Additionally, the Common Weakness Enumeration<sup>4</sup> was considered as a background source, linking weaknesses identified by SonarRules and providing additional information. Finally, the article “Mining frequent bug-fix code changes” [32] also contributed to a code perfume definition.

#### A. Java Standard Library Code Perfumes

Code perfumes in this category emphasise the importance of understanding concepts from the Java standard library, which beginners should encounter early in their coding journey [31].

**Clone blueprint:** This code perfume focuses on implementing the `java.lang.Cloneable` interface to provide a copy functionality. While it may not be the ideal approach, it is a valid option if done correctly. The code perfume emphasises the importance of consistently calling `super.clone` in `clone` overrides in an inheritance hierarchy, to maintain the correct runtime typing of the cloned object. The code perfume aligns with the Common Weakness Enumeration (CWE-580) and the corresponding SonarSource rule (RSPEC-1182).

```
class DisplayData implements Cloneable {
    @Override
    public synchronized DisplayData clone() {
        DisplayData displayData;
        try {
            displayData = (DisplayData) super.clone();
        } catch (CloneNotSupportedException e) {
            throw new Error(e);
        }
        // Deep copy
        return displayData;
    }
}
```

<sup>4</sup><https://cwe.mitre.org/index.html>, last accessed 08.06.2024

**Equals blueprint:** Overriding the `equals` method in the `java.lang.Object` class requires adherence to a specific contract, implementing an equivalence relation between instances of the class. To simplify the coding process and reduce the likelihood of errors, Joshua Bloch provides a guideline [31]. This code perfume focuses on the first two steps of Bloch’s guideline, checking object references and type correctness, as the comparison of class fields can vary.

```
@Override
public boolean equals(Object o) {
    if (o == this) {
        return true;
    }
    if (!(o instanceof Pawn)) {
        return false;
    }
    // Field checks
}
```

**Iterator next() follows the contract:** This code perfume focuses on classes that implement the `java.util.Iterator` interface in Java. This interface allows for iterating over data structures, such as lists. The key methods for iteration are `hasNext` and `next`, with the latter having a contractual requirement to throw a `NoSuchElementException` when called without a remaining element in the iteration. This behaviour is expected by developers using iterators, and omitting it can lead to unexpected errors. This code perfume checks whether a class implementing the `Iterator` interface adheres to the necessary step of throwing a `NoSuchElementException` in the `next` method. It is inspired by the rule from SonarSource (RSPEC-2272), which considers the absence of this behaviour a bug.

```
public final class Range
    implements Iterator<Integer> {
    @Override
    public boolean hasNext() {
        return contains(start);
    }
    @Override
    public Integer next() {
        if (!hasNext()) {
            throw new NoSuchElementException();
        }
        // Iterator logic
    }
}
```

**Override compareTo with equals:** The `java.lang.Comparable` interface includes the `compareTo` method for defining a natural ordering between instances of the class. The method’s contract suggests that if two objects are equal according to the `equals` method, they should also be equal according to the `compareTo` method, and vice versa. While this is only a recommendation, it is considered good practice to override both methods. The code perfume serves as a solution pattern to the corresponding SonarSource rule (RSPEC-1210) checking the presence one of these methods.

```
class Command implements Comparable<Command> {
    @Override
    public int compareTo(Command command) { ... }
```

TABLE I: Categories and types of code perfumes

Category	ID	Code Perfume	Type
Java Standard Library	1	Clone blueprint	solution pattern, contract or convention
	2	Equals blueprint	contract or convention, best practice
	3	Iterator next() follows the contract	contract or convention
	4	Override compareTo with equals	solution pattern, best practice
	5	Override equals of superclass	solution pattern
	6	Paired equals and hashCode	contract or convention, solution pattern
	7	Use optimized collections for Enums	best practice, solution pattern
Java language features	8	'assert' in private method	solution pattern, best practice
	9	At least X varargs	solution pattern
	10	Defensive default case	solution pattern
	11	Pattern matching with 'instanceof'	solution pattern
	12	Resource management in try-catch	best practice, solution pattern
Unit Testing Practices	13	Synchronize accessors in pairs	solution pattern
	14	JUnit 5 tests can be package-private	solution pattern
	15	Single method call when testing for runtime exceptions	solution pattern
Design Patterns	16	Builder pattern	best practice
	17	Singleton pattern	best practice
Others	18	Copy constructor	best practice
	19	Defensive null check	best practice, solution pattern
	20	No utility instantiation	solution pattern, best practice

```

@Override
public boolean equals(Object o) { ... }
}

```

**Override equals of superclass:** This code perfume addresses a common issue in inheritance hierarchies where subclasses add fields to those already defined in their superclasses. If a superclass overrides the `equals` method, but the subclass fails to override it, there is a potential mistake. SonarSource (RSPEC-2160) identifies this, as it may lead to unexpected behaviour where the additional fields in the subclass are ignored by the `equals` method.

```

public class Ancestor {
    @Override
    public boolean equals(Object o) { ... }
}
public class Child extends Ancestor {
    int additionalField;
    @Override
    public boolean equals(Object o) { ... }
}

```

**Paired equals and hashCode:** This code perfume highlights the importance of maintaining consistency between the `equals` and `hashCode` methods in Java. While classes can override these methods independently, it is considered best practice to override both or neither to avoid violating the contract they form. The contract, particularly for `hashCode`, states that objects deemed equal by the `equals` method should have the same `hashCode`. Although the contract, also denoted by SonarSource (RSPEC-1206) and CWE (CWE-581), allows distinct objects to have the same `hashCode`, it is recommended to override both methods together for consistent behaviour and improved performance in hash tables.

```

public class SudokuBoard implements Board {
    @Override
    public boolean equals(Object other) { ... }

    @Override
    public int hashCode() { ... }
}

```

**Use optimised collections for Enums:** This code perfume focuses on optimising memory usage and execution speed in Java applications, particularly when dealing with Enum types. It encourages the use of optimised classes provided by the Java standard library's Collections API for Enums. The recommendation to use these optimised collections for Enums is emphasised by Joshua Bloch [31] and two SonarSource rules (RSPEC-1640 and RSPEC-1641).

```

public class KalahBoard implements Board {
    private final EnumMap<Player, PlayerPits>
        playerPits;
    public KalahBoard( ... ) {
        this.playerPits =
            new EnumMap<>(Player.class);
        // Constructor code
    }
}

```

#### B. Java Language Feature Code Perfumes

Code perfumes in this category encourage the use of language features, idioms, and keywords in Java coding practices, with a focus on defensive programming [33] and addressing code smells or bug patterns [34].

**'assert' in private method:** While the `assert` keyword is useful for validating parameters, it should be avoided in public methods because assertions must be explicitly enabled at the JVM level. Best practice involves using `assert` in

private methods during development for parameter validation. To enhance clarity and to ease error identification, the `assert` expression should be accompanied by a message. This aligns with SonarSource's rule (RSPEC-4274) against using `asserts` to check public method parameters, and is also highlighted in "Effective Java" [31].

```
private boolean isMovePossible(int row,
    int diag, Color color) {
    assert isValidPosition(row, diag)
        : "Invalid coordinates";
    // Method logic
}
```

**At least X varargs:** This code perfume addresses the use of varargs in Java methods, emphasising a more efficient approach to set a lower bound on the number of parameters. Instead of checking the number of parameters within the method's body resulting in unnecessary code, it encourages declaring a specific number of parameters before the varargs. This approach, recommended by Joshua Bloch [31], allows for compile-time validation of the minimum argument count, improving efficiency compared to runtime checks.

```
public static boolean isWithinCharset(
    int charsetSize, int... symbols) {
    return symbols != null
        && Arrays.stream(symbols)
            .allMatch(i -> i >= 0 && i < charsetSize);
}
```

**Defensive default case:** This code perfume underscores the importance of using `switch` statements effectively as a cleaner alternative to chaining `if`-statements. The `switch` construct allows handling every possibility of the `switch`-variable, with the `default` case serving as a safety net to address unforeseen situations. Including a `default`-case at the end of a `switch` statement demonstrates defensive programming, reducing the risk of unexpected behaviour due to overlooked cases as reported by SonarSource (RSPEC-131 and CWE (CWE-478)).

```
switch (firstLetter) {
    case 'h':
        cmdHelp(token);
        break;
    // More cases
    default:
        printError("Invalid Command!");
        break;
}
```

**Pattern matching with 'instanceof':** This code perfume promotes the use of the `instanceof` operator in conjunction with pattern matching, introduced in Java 16. While the `instanceof` operator traditionally required manual casting, pattern matching now allows for a concise type-check and cast in a single `instanceof` statement. This enhances clarity and readability, as promoted by SonarSource (RSPEC-6201).

```
if (copy instanceof Game gameCopy) {
    gameCopy.cells = cells.clone();
}
```

**Resource management in try-catch:** This code perfume encourages the use of the try-with-resources statement in Java to ensure the proper closing of resources within a try-catch block. The try-with-resources statement is considered a best

practice inspired by SonarSource (RSPEC-2093) to prevent accidental resource leaks, especially for beginners working with file reading and I/O streams.

```
try (BufferedReader reader = new BufferedReader(
    new FileReader(file))) {
    Line line = readLine(reader, 0);
    // Method logic
}
```

**Synchronize accessors in pairs:** This code perfume encourages careful handling of parallelisation, emphasising the use of the `synchronized` keyword to synchronise specific parts of the execution flow and prevent race conditions. It particularly focuses on scenarios involving accessor methods (getters and setters) for class fields. The code perfume rewards the practice of synchronising both accessors, addressing a potential bug as identified by SonarSource (RSPEC-2886).

```
synchronized void setCell(int rowNr, int colNr,
    int number) { ... }

synchronized int getCell(int rowNr, int colNr)
{ ... }
```

#### C. Unit Testing Code Perfumes

Code perfumes in this category recognise the standard practice of unit testing [35] and the need for high-quality test code to avoid maintainability issues [36], [37], [38].

**JUnit 5 tests can be package-private:** This code perfume advocates for adhering to the principle of operating with the least required privileges in Java coding. Specifically, it encourages developers to choose the narrowest scope possible for class visibility, including unit tests. With JUnit 5, both tests and test classes do not have to be public anymore, making them package-private is sufficient for JUnit to detect and execute them; inspired by SonarSource (RSPEC-5786).

```
class PerfumeTest {
    @Test
    void someTest() { ... }
}
```

**Single method call when testing for runtime exceptions:** This code perfume emphasises the importance of testing code for expected `RuntimeExceptions` using frameworks like JUnit and AssertJ. It discourages chaining multiple method calls within the `assertThrows` or `assertThatThrownBy` methods, as this can obscure which specific method call caused the expected exception or whether any call did at all. The code perfume suggests using only a single method call in these methods for improved code clarity and consistency as taken from SonarSource (RSPEC-5778).

```
@Test
void someTest() {
    assertThrows(Exception.class,
        () -> callMethod());
}
```

#### D. Design Pattern Code Perfumes

Code perfumes in this category acknowledge the value of design patterns in advanced programming education and their contribution to well-formed code [39], [40], [41].

**Builder pattern:** This code perfume underscores the benefits of using the Builder pattern to address the challenge of classes with numerous optional fields. Instead of creating an excessive number of constructor overloads, the Builder pattern involves an inner class responsible for constructing instances of the outer class. This approach is more flexible, allowing easier extension of the builder and the creation of multiple instances using the same builder. Joshua Bloch [31] recommends considering the Builder pattern when a class's constructor would require four or more parameters.

```
public class ToBuild {
    private ToBuild(Builder builder) { ... }
    public static class Builder {
        public ToBuild build() {
            return new ToBuild(this);
        }
    }
}
```

**Singleton pattern:** This code perfume focuses on the Singleton pattern for enforcing the existence of exactly one instance of a class. Joshua Bloch [31] describes three main ways to implement the Singleton pattern: using a public instance field, a public static instance factory method, or defining the class as an enum type with a single constant. Each approach has its advantages, and all are considered valid.

```
static final class Controller {
    private static Controller instance;
    private Controller() { }
    public static Controller getInstance() {
        return instance;
    }
}
```

#### E. Other Code Perfumes

Finally, some code perfumes fit in no other category.

**Copy constructor:** This code perfume considers object copying in Java, emphasising the pitfalls associated with the Cloneable interface, as highlighted by Joshua Bloch [31]. A copy constructor is recommended as a clean solution that takes an object of the same class as its parameter and performs a deep copy of all fields, including mutable non-primitive ones. This approach ensures the desired runtime type and is less error-prone than implementing Cloneable.

```
public Tape(Tape clone) {
    content = new LinkedList<>(clone.content);
    headPosition = clone.headPosition;
    move(Direction.NOTHING);
}
```

**Defensive null check:** This code perfume underscores the importance of caution when declaring public methods that deal with non-primitive parameter types. The risk of bugs and unexpected behaviour is increased when null values are passed as parameters. To prevent unexpected NullPointerExceptions at runtime, the code perfume recommends checking non-primitive parameters for non-null values or using annotations like `@NotNull` or `@Nonnull` to specify that null values are not allowed. Alternatively, if the method permits null values for certain parameters, using an `@Nullable` annotation is advised by Joshua Bloch [31].

```
public static void callPrint(
    TuringMachine turingMachine) {
    if (turingMachine != null) {
        System.out.println(turingMachine.print());
    } else {
        errorMsg("No Turing machine initialized.");
    }
}
```

**No utility instantiation:** This code perfume emphasises the definition of a private constructor for utility classes containing only static methods. Since utility classes do not need to be instantiated and thus need no implicit default constructor, having a single private constructor demonstrates an understanding of Java mechanics and object orientation as described by SonarSource (RSPEC-1118).

```
public final class Shell {
    private Shell() { }
    private static void process(String[] strings)
    { ... }
    private static void add(String[] strings)
    { ... }
}
```

#### F. Perfumator: A Tool to Detect Code Perfumes in Java Code

The *Perfumator* is a static analysis tool implemented in Java 17, with its primary purpose being the detection of code perfumes in abstract syntax trees (ASTs). Its functionality involves two main steps: defining static information in a JSON file and implementing a `Detector` class that recognises specific patterns in the AST. JavaParser is utilized by the *Perfumator* for parsing ASTs and resolving symbols. The tool offers a command-line interface featuring options like input path, output format, batch size, specifying dependencies as well as support for internationalization.

The *Perfumator* identifies instances of code perfumes in designated input files or directories and generates serialised output in JSON format containing detailed information. To enhance accuracy, the tool allows ignoring specific files and directories like the build directory during analysis. The *Perfumator* is designed to be easily extendable with new code perfumes and holds the potential for integration into other tools.

## IV. EVALUATION

To gain an initial understanding of the prevalence of the code perfumes defined in this paper, and how they relate to correctness, we applied the *Perfumator* to a large dataset of programming assignments and student solutions. This allows us to address the following research questions:

- **RQ1:** How common are code perfumes in Java?
- **RQ2:** Are code perfumes related to correctness?

#### A. Experimental Setup

In order to study code perfumes, we use a dataset derived from the “Programming II” course taught every semester at the University of Passau. This course focuses on intermediate/advanced Java programming, covering topics such as Java Swing and parallelism. The coursework consists of four programming assignments. Although the specific tasks vary

TABLE II: Assignment submissions and size

Task	Name	Total LoC	Submissions	Average LoC
0	Practice	133931	359	373
1	Algorithms	442450	809	547
2	CLI Board Game	659318	607	1086
3	GUI Board Game	609915	560	1089

between different instances of the course, they are always variations of the same type of program, and usually represent the implementation of a board game like Abalone<sup>5</sup> or Mastermind<sup>6</sup>. Due to the similarity of the solution code, we merge different variants of the same task in our dataset. The tasks include:

- Task 0: Exercise task for familiarity with the automatic grading system
- Task 1: Mathematical problem-solving to acquaint students with organising programs and creating complex algorithms
- Task 2: Modelling the board game with a command-line interface
- Task 3: Implementing a user interface for the previously modeled game

The first assignment only serves for practice, while the remaining assignments determine the overall grade. Until the submission deadline, students can re-submit as often as they wish, receiving feedback from a set of public tests. After the deadline, functionality is assessed using a set of secret tests and normalised to the range 1 (all tests pass) to 6 (no tests pass). Additionally, each submission is graded manually on readability considering coding styles, code structure, and variable/method naming on a range of 1 to 6. The overall grade is a weighted sum of functionality and readability ranging from 1 (best) to 5 (worst) with intermediate steps of .3 and .7.

We use data from this course between the winter semester of 2015/2016 to the winter semester of 2022/2023, covering 15 iterations of the course. Students who registered but did not submit a task were excluded from the analysis. This resulted in a dataset of 816 students, acknowledging that some students may have participated more than once to pass the course. Table II shows that less than half of the students submitted Task 0, which was optional. The highest number of submissions was for Task 1, the first graded task, with subsequent tasks seeing declining submission rates. The lines of code (LoC) increased with each task, with Tasks 2 and 3 having almost identical LoC counts.

To understand the limitations and applicability of our code perfumes, we also consider the Progpedia dataset [42] consisting of 16 programming assignments for undergraduate students. The dataset consists of 4362 student solutions. The assignments are very basic and cover topics from reading user input from the console to basic algorithms like Kruskal [43].

1) *RQ1: How common are code perfumes in Java?:* To answer this research question, we execute the *Perfumator* on all student submissions for both datasets and extract the number of code perfumes per category.

<sup>5</sup>[https://en.wikipedia.org/wiki/Abalone\\_\(board\\_game\)](https://en.wikipedia.org/wiki/Abalone_(board_game)), last accessed 08.06.2024

<sup>6</sup>[https://en.wikipedia.org/wiki/Mastermind\\_\(board\\_game\)](https://en.wikipedia.org/wiki/Mastermind_(board_game)), last accessed 08.06.2024

2) *RQ2: Are code perfumes related to correctness?:* To answer this research question, we compare the number of code perfumes with other metrics of their solutions, such as the grade on functionality, the grade on readability, the overall grades for the course, and the number of lines of code (LoC), assessed using *CLOC*<sup>7</sup>. To contrast the insights gained from positive code aspects such as code perfumes, we also examine negative aspects by extracting the number of code smells using *SonarLint*<sup>2</sup>. Subsequently, we use Pearson correlation [44] to determine potential correlations between these metrics.

### B. Threats to Validity

*Threats to internal validity* may result from task selection, grading criteria, and inconsistencies in both automatic and manual grading processes. However, the grading process remained consistent over the years with the same supervisor and the same grading criteria. Moreover, the act of testing itself might influence students' subsequent performance, and external factors like changes in course structure or students' natural skill progression could also impact the outcomes. The *Perfumator* might not accurately detect all perfumes, but we minimized this issue by verifying the results beforehand.

*Threats to external validity* exist because findings might not generalise beyond the specific student population or programming course and university. Additionally, the tasks may not accurately mirror real-world programming scenarios, and changes over time in technology, education methods, or student demographics could affect the results.

*Threats to construct validity* may result from the tools utilised for assessment, which may not fully capture the intended variables or could introduce errors. All students had the possibility to analyse their graded results and discuss them with the supervisor, therefore reducing this threat. Maintaining consistency in task specifics and evaluation across different course iterations potentially affects treatment effects.

### C. RQ1: How Common are Code Perfumes in Java?

Table III lists the overall number of code perfumes found, their distribution across tasks, and the percentage of submissions containing them. In total, the *Perfumator* identified 8598 instances of perfumes in the dataset. Only a few of the code perfumes are not found, which is due to the nature of the tasks: The absence of the “override equals of superclass” code perfume (ID 5) is expected, given the relatively simple project structures that may not involve extensive inheritance hierarchies. Code perfumes related to testing (IDs 14 and 15) are unsurprisingly absent, considering the absence of unit testing in the course. Similarly, design patterns (IDs 16 and 17) are infrequently used, possibly due to students' limited awareness of builder and singleton patterns or the tasks not necessitating their implementation. The copy constructor code perfume (ID 18) sees limited use, as most students opt for the cloning functionality (ID 1). However, all other code perfumes can be found frequently in the dataset, suggesting that they are relevant and suitable for the type of code we are targeting.

<sup>7</sup><https://github.com/AlDanial/cloc>, last accessed 08.06.2024

TABLE III: Code Perfumes overall and per task

ID	Total	Submissions	Task 0	Task 1	Task 2	Task 3
1	196	4.0%	0	3	97	96
2	17	0.3%	0	7	5	5
3	2	0.0%	0	0	1	1
4	281	5.8%	0	254	12	15
5	0	—	0	0	0	0
6	419	8.3%	0	259	79	81
7	10	0.1%	0	0	5	5
8	520	1.7%	1	6	257	256
9	4	0.0%	0	1	3	0
10	3656	43.3%	426	871	1346	1013
11	21	0.3%	0	2	2	17
12	134	3.0%	2	80	37	15
13	9	0.1%	0	0	0	9
14	0	—	0	0	0	0
15	0	—	0	0	0	0
16	0	—	0	0	0	0
17	17	0.2%	0	0	6	11
18	21	0.5%	0	5	7	9
19	1048	11.5%	125	271	250	402
20	2243	43.0%	4303	851	671	418

The distribution of code perfumes reveals insights into the coding behaviour of the students. For example, defensive programming practices are very prevalent among students and these assignments. Three specific code perfumes (IDs 10, 19, and 20) collectively constitute over 80% of all detected instances, indicating a strong emphasis on defensive programming. Notably, the “assert in private method” code perfume (ID 8) is frequently encountered, along with the “defensive null check” code perfume (ID 19), suggesting a preference for null checks in public methods over assertions.

The code perfume occurrences also suggest adherence to certain Java language conventions, particularly in cases where students override the `equals` method alongside either `hashCode` or `compareTo` (IDs 4 and 6). However, the students notably deviate from the common `equals` blueprint by not consistently checking object references and type correctness. An intriguing observation is also that students rarely employ synchronized accessors in pairs (ID 13), a crucial aspect introduced in the final assignment of each course. This oversight could lead to runtime exceptions and deadlocks in their implemented games. Optimised collections for enums (ID 7) are infrequently used, indicating potential unawareness of these specialised collections. The same may apply to the “at least X varargs” code perfume (ID 9), while the “pattern matching with instanceof” code perfume (ID 11) reflects the limited adoption due to its recent functionality release in 2021. This raises the question whether the inclusion of code perfumes into student feedback could help improving such deficiencies.

It is noteworthy that the frequency of finding a code perfume does not necessarily imply even distribution across all submissions. For instance, although code perfume ID 20 is encountered twice as often as code perfume ID 19, the former appears in nearly four times as many submissions as the latter (see Table III). Similarly, code perfumes 6 and 8 exhibit a similar trend, with code perfume 6 appearing in nearly five times more submissions than code perfume 8. Consequently,

some students have internalised aspects represented by code perfumes and use them frequently, while those with fewer occurrences may still be in the process of adopting these constructs. It is conceivable that positive reinforcement by reporting code perfumes could support this learning process.

To better understand the limits of the code perfumes defined in this paper, we contrast this data with the ProgPedia dataset of introductory assignments, where we only found a total of 14 code perfume instances (IDs 10, 18, and 19) on 4362 student submissions. This can be attributed to the very simplistic nature of those assignments, they only consist of one file and mostly can be done in 30 to 60 LoC. While this shows that some students are aware of defensive programming this early in their programming education, more code perfumes targetting earlier, simpler stages of coding are clearly needed to fittingly evaluate students in a CS1/Algorithmics class, while our code perfumes are more fitted for students in a CS2/Software Engineering class, as shown above.

**Summary (RQ1):** The *Perfumator* identified 8598 instances of code perfumes, highlighting a significant emphasis on defensive programming practices among students, while other conventions are not as commonly embraced. Our code perfumes target intermediate learners covering design principles; more code perfumes are needed to cover beginners’ code.

#### D. RQ2: Are Code Perfumes Related to Correctness?

Figure 1 visualises the correlations between the metrics considered. According to Pearson correlation [44], there is a weak negative correlation between the number of code perfumes and the functionality (Fig. 1a) and readability (Fig. 1b) assigned to students assignments. Additionally, there is a moderate negative correlation between the number of code perfumes and the grades (Fig. 1c) received by the students. Since the best grade is “1”, the negative correlations suggest that code perfumes can serve as an indicator for correctness as well as code quality, and thus represent a valuable tool for educators.

However, we can generally also see that LoC are correlated with functionality (Fig. 1f), readability (Fig. 1g), and grades (Fig. 1h), as more complete solutions tend to consist of more code. When normalising code perfumes per LoC, we find no correlations between code perfumes per LoC and functionality (0.01), readability (-0.04), and grades (0.06). An explanation for this is that our 20 code perfumes are too few. While code perfumes can inform tutors about their students and aspects they perform well on, for code perfumes to serve as a reliable predictor for student performance, more code perfumes will need to be defined that cover more aspects of functionality and readability. The immediate application of code perfumes for educators thus for now primarily lies in observing which aspects of code are handled well by students, or identifying students that are lacking understanding of certain concepts.

The complementary nature of code smells and code perfumes is indicated by the substantially weaker correlations between code smells and functionality (Fig. 1j), readability (Fig. 1k), and grades (Fig. 1l), even though more code also

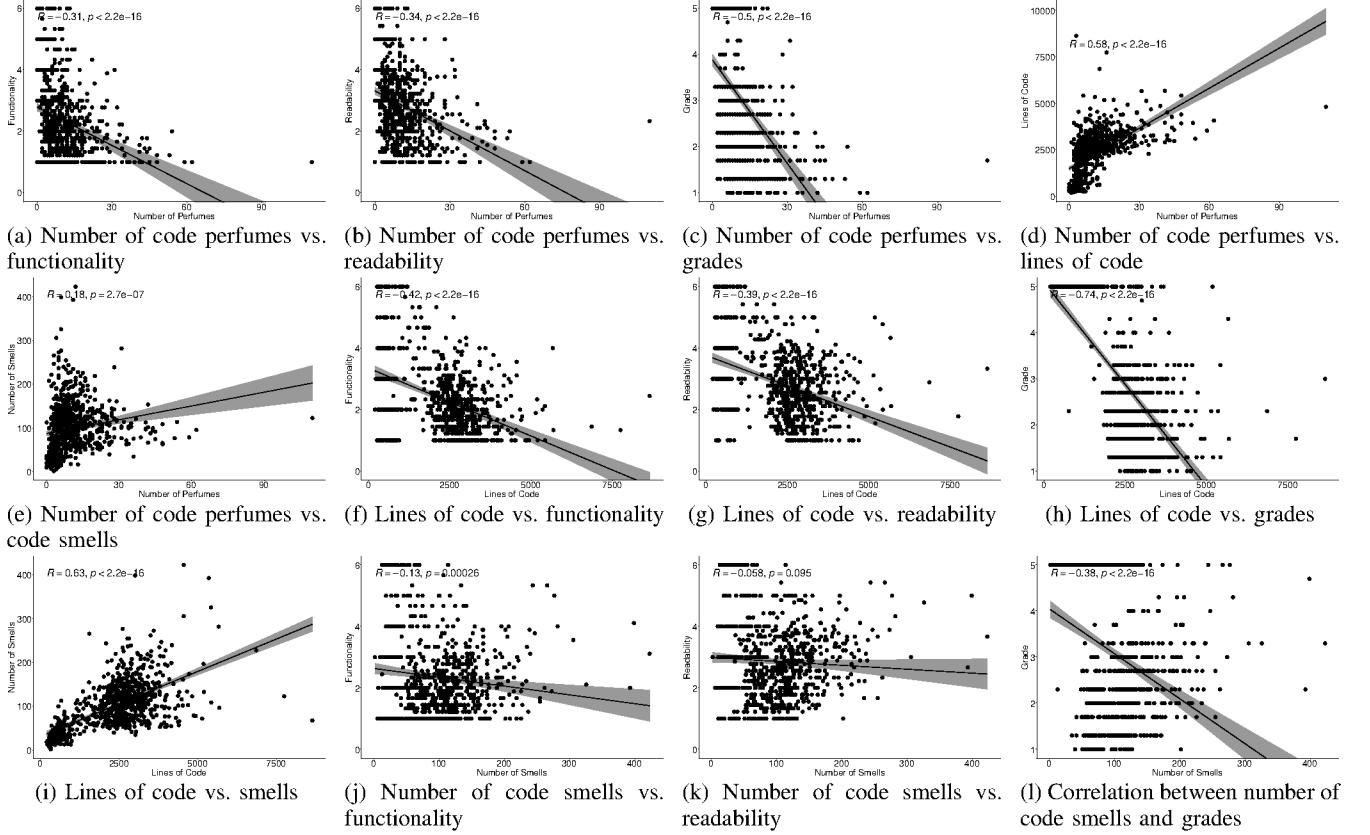


Fig. 1: Correlations between number of code perfumes, code smells, line of code, functionality, readability, and grades

leads to more code smells (Fig. 1i). The most common code smells are related to logging, cognitive complexity, optimising conditionals, multi-threading and code duplication. The analyses on code smells are influenced by the large differences between the numbers of code perfumes and code smells, as SonarLint defines 677 rules [45]. On the one hand, this reinforces the conclusion that more code perfumes are needed. On the other hand, it also suggests that neither mechanism on its own can be used for reliable grading, but the combination of negative and positive aspects provides valuable information. Thus, classic linters and the perfumator could be used in combination to create a more holistic view of the learners project helping educators support and assess their learners.

**Summary (RQ2):** The presence of code perfumes is associated with improved readability, functionality, and higher grades, and therefore represent a valuable tool to inform educators about their students progress. However, the vast difference in the number of existing code smells and code perfumes suggests future work on defining more code perfumes.

## V. CONCLUSIONS

Program analysis can point out problems in code, but they usually do not provide encouraging, positive feedback. In this paper, we therefore introduced the concept of code perfumes for Java, and presented 20 code perfumes to provide

learners with constructive feedback. Our analysis indicates that many of these code perfumes are common in practice, and we find a link between the presence of code perfumes and program correctness, highlighting their potential to aid teachers' understanding of student learning progress.

While these initial findings are encouraging, our catalogue of 20 code perfumes is by no means complete, and we anticipate future work to result in many additional code perfumes. Investigating how to integrate these in the learning process will be similarly important. For example, code perfumes could be annotated directly in Integrated Development Environments (IDEs) or in gamified environments such as Gamekins [46]. Educators may need different ways to interface with code perfumes, for example by incorporating them as a metric in grading and learning systems such as ArTEMiS [15]. Finally, although our motivation originally was to support learners, there is no reason why code perfumes could not also be used to encourage professional programmers.

The *Perfumator* is available at:

<https://github.com/se2p/perfumator>

## ACKNOWLEDGEMENTS

This work is supported by the DFG under grant FR 2955/2-1, “QuestWare: Gamifying the Quest for Software Tests”. Thanks to Jakob Silbereisen for his contributions to the *Perfumator*.

## REFERENCES

- [1] W. Conceição, M. Holanda, F. Macedo, E. Ishikawa, V. T. Nunes, and D. Da Silva, "Automatic feedback in the teaching of programming in undergraduate courses: a literature mapping," in *2022 IEEE Frontiers in Education Conference (FIE)*, 2022.
- [2] S. C. dos Santos, P. C. A. R. Tedesco, M. Borba, and M. Brito, "Innovative approaches in teaching programming: A systematic literature review," in *International Conference on Computer Supported Education*, 2020.
- [3] T. Effenberger and R. Pelánek, "Code quality defects across introductory programming topics," in *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education - Volume 1*, SIGCSE 2022, Association for Computing Machinery, 2022.
- [4] B. Wisniewski, K. Zierer, and J. Hattie, "The power of feedback revisited: A meta-analysis of educational feedback research," *Frontiers in Psychology*, 2020.
- [5] J. Hattie, *Visible Learning: A Synthesis of Over 800 Meta-Analyses Relating to Achievement*. 01 2009.
- [6] F. Obermüller, L. Bloch, L. Greifenstein, U. Heuer, and G. Fraser, *Code Perfumes: Reporting Good Code to Encourage Learners*. Association for Computing Machinery, 2021.
- [7] M. Fowler, *Refactoring: Improving the Design of Existing Code*. Boston, MA, USA: Addison-Wesley, 1999.
- [8] F. Hermans and E. Aivaloglou, "Do code smells hamper novice programming? a controlled experiment on scratch programs," in *2016 IEEE 24th International Conference on Program Comprehension (ICPC)*, 2016.
- [9] V. Gutmann, E. Starke, and T. Michaeli, "Investigating code smells in k-12 students' programming projects: Impact on comprehensibility and modifiability," 2022.
- [10] D. Hovemeyer and W. Pugh, "Finding bugs is easy," *SIGPLAN Not.*, 2004.
- [11] A. Swidan, F. Hermans, and M. Smit, "Programming misconceptions for school students," in *Proceedings of the 2018 ACM Conference on International Computing Education Research*, ICER '18, Association for Computing Machinery, 2018.
- [12] C. Frädrich, F. Obermüller, N. Körber, U. Heuer, and G. Fraser, "Common bugs in scratch programs," in *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE '20, Association for Computing Machinery, 2020.
- [13] J. Sorva, *Misconceptions and the Beginner Programmer*. 2018.
- [14] J. Jansen, A. Oprescu, and M. Bruntink, "The impact of automated code quality feedback in programming education," in *Post-proceedings of the Tenth Seminar on Advanced Techniques and Tools for Software Evolution (SATToSE)*, 2017.
- [15] S. Krusche and A. Seitz, "Artemis: An automatic assessment management system for interactive learning," in *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, SIGCSE 2018, ACM, 2018.
- [16] A. Collins and M. Kapur, *Cognitive Apprenticeship*, p. 109–127. Cambridge Handbooks in Psychology, Cambridge University Press, 2 ed., 2014.
- [17] T. Michaeli and R. Romeike, "Current status and perspectives of debugging in the k12 classroom: A qualitative study," in *2019 IEEE Global Engineering Education Conference (EDUCON)*, IEEE, 2019.
- [18] A. Yadav, S. Gretter, S. Hambrusch, and P. Sands, "Expanding computer science education in schools: understanding teacher experiences and challenges," *Computer Science Education*, 2016.
- [19] S. Sentance and A. Csizmadia, "Computing in the curriculum: Challenges and strategies from a teacher's perspective," *Education and Information Technologies*, 2017.
- [20] S. DePasque and E. Tricomi, "Effects of intrinsic motivation on feedback processing during learning," *NeuroImage*, 2015.
- [21] V. J. Marin, T. Pereira, S. Sridharan, and C. R. Rivero, "Automated personalized feedback in introductory java programming moocs," in *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*, 2017.
- [22] S. Arai, K. Sakamoto, H. Washizaki, and Y. Fukazawa, "A gamified tool for motivating developers to remove warnings of bug pattern tools," in *2014 6th International Workshop on Empirical Software Engineering in Practice*, 2014.
- [23] Z. Zhan, L. He, Y. Tong, X. Liang, S. Guo, and X. Lan, "The effectiveness of gamification in programming education: Evidence from a meta-analysis," *Computers and Education: Artificial Intelligence*, 2022.
- [24] J. Maloney, M. Resnick, N. Rusk, B. Silverman, and E. Eastmond, "The scratch programming language and environment," *ACM Transactions on Computing Education (TOCE)*, 2010.
- [25] J. Moreno-León, G. Robles, and M. Román-González, "Dr. scratch: Automatic analysis of scratch projects to assess and foster computational thinking," *RED-Revista de Educación a Distancia*, 2015.
- [26] G. Fraser, U. Heuer, N. Körber, F. Obermüller, and E. Wasmeier, "Litterbox: A linter for scratch programs," in *43rd IEEE/ACM International Conference on Software Engineering: Software Engineering Education and Training, ICSE (SEET) 2021*, IEEE, 2021.
- [27] J. Börstler, H. Störrle, D. Toll, J. van Assema, R. Duran, S. Hooshangi, J. Jeuring, H. Keuning, C. Kleiner, and B. K. MacKellar, "'i know it when I see it': Perceptions of code quality," in *Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE 2017, 2017.
- [28] F. Obermüller, L. Bloch, L. Greifenstein, U. Heuer, and G. Fraser, "Code perfumes: Reporting good code to encourage learners," in *WiPSCE '21: The 16th Workshop in Primary and Secondary Computing Education*, ACM, 2021.
- [29] L. Porter and B. Simon, "Retaining nearly one-third more majors with a trio of instructional best practices in CS1," in *The 44th ACM Technical Symposium on Computer Science Education*, SIGCSE 2013, ACM, 2013.
- [30] D. Deveaux, P. Frison, and J. Jézéquel, "Increase software trustability with self-testable classes in java," in *13th Australian Software Engineering Conference (ASWEC 2001)*, IEEE Computer Society, 2001.
- [31] J. J. Bloch, *Effective Java, 3rd Edition*. The Java series ... from the source, Addison-Wesley, 2017.
- [32] H. Osman, M. Lungu, and O. Nierstrasz, "Mining frequent bug-fix code changes," in *2014 Software Evolution Week - IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering, CSMR-WCRE 2014*, IEEE Computer Society, 2014.
- [33] J. K. Teto, R. Bearden, and D. C. Lo, "The impact of defensive programming on I/O cybersecurity attacks," in *Proceedings of the 2017 ACM Southeast Regional Conference*, 2017.
- [34] P. Weill-Tessier, A. L. Costache, and N. C. C. Brown, "Usage of the java language by novices over time: Implications for tool and language design," in *SIGCSE '21: The 52nd ACM Technical Symposium on Computer Science Education*, ACM, 2021.
- [35] E. Dako and G. Fraser, "A survey on unit testing practices and problems," in *25th IEEE International Symposium on Software Reliability Engineering, ISSRE 2014*, 2014.
- [36] R. Ramler, M. Moser, and J. Pichler, "Automated static analysis of unit test code," in *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, IEEE, 2016.
- [37] R. C. Martin, *The clean coder: a code of conduct for professional programmers*. Pearson Education, 2011.
- [38] A. Van Deursen, L. Moonen, A. Van Den Bergh, and G. Kok, "Refactoring test code," in *Proceedings of the 2nd international conference on extreme programming and flexible processes in software engineering (XP2001)*, Citeseer, 2001.
- [39] R. Helm, R. E. Johnson, E. Gamma, and J. Vlissides, *Design patterns: Elements of reusable object-oriented software*. Braille Jymico Incorporated Quebec, 2000.
- [40] R. Fojtik, "Design patterns in the teaching of programming," *Procedia-Social and Behavioral Sciences*, 2014.
- [41] A. Blewitt, A. Bundy, and I. Stark, "Automatic verification of design patterns in java," in *20th IEEE/ACM International Conference on Automated Software Engineering (ASE 2005)*, ACM, 2005.
- [42] J. C. Paiva, J. P. Leal, and Álvaro Figueira, "Progpedias: Collection of source-code submitted to introductory programming assignments," *Data in Brief*, 2023.
- [43] J. B. Kruskal, "On the shortest spanning subtree of a graph and the traveling salesman problem," *Proceedings of the American Mathematical society*, 1956.
- [44] I. Cohen, Y. Huang, J. Chen, J. Benesty, J. Benesty, J. Chen, Y. Huang, and I. Cohen, "Pearson correlation coefficient," *Noise reduction in speech processing*, 2009.
- [45] SonarSource, "Unique rules to find bugs, vulnerabilities, security hotspots, and code smells in your java code." <https://rules.sonarsource.com/java/>, 2024. Accessed: 08.06.2024.
- [46] P. Straubinger and G. Fraser, "Gamifying a software testing course with continuous integration," in *46th IEEE/ACM International Conference on Software Engineering, ICSE 2024*, 2024.

# Technology Acquisition Plans to Foster Supply Chain Risk Management Learning Outcomes in Project-Based Software Development Courses

Bastian Tenbergen

*Department of Computer Science  
State University of New York at Oswego  
Oswego, NY, United States  
[bastian.tenbergen@oswego.edu](mailto:bastian.tenbergen@oswego.edu)*

Nancy R. Mead

*SEI Fellow, Carnegie Mellon University  
IEEE Life Fellow  
Pittsburgh, PA, United States  
[nrmcmu@gmail.com](mailto:nrmcmu@gmail.com)*

**Abstract**—From theory and technical skills to self-management and interpersonal skills, and to role-specific knowledge such as DevOps or Requirements Engineering, students struggle to become proficient in all aspects to Software Engineering before graduating, while instructors struggle to prepare students for industry. Project-based instruction has become a de-facto standard to teach skills and knowledge. Many project-based courses focus on implementation, wherein students tend to adopt third-party technologies without critical reflection. This leads to insufficient knowledge about supply chain risk management (SCRM). The result is that students are inadequately prepared to handle risks to project success due to third-party acquisitions, including usage rights, introduction of security vulnerabilities, or their ability to identify and mitigate acquisition risks. In this paper, we present a capstone-style project-based course to deliver an industry-realistic and teamwork-centric software development approach. We propose incorporating Technology Acquisition Plans, which are based on an existing SCRM curriculum, and add acquisition planning, bidding, and risk monitoring activities to the project and development activities. Empirical evaluation and classroom observations show that while students' ability to justify the need for third-party acquisitions remains the same, self-reported ability to evaluate and bid on alternatives is improved. Furthermore, statistically significant improvement can be seen in areas pertaining to risk monitoring, problem detection, and problem resolution.

**Keywords**—software engineering education, supply chain risk management, SCRM, software acquisition, team management, project-based learning, technology acquisition plans

## I. INTRODUCTION

Software Engineering education (SEE) has taken a strong, positive turn in the past two decades [1], [2]. Since Mary Shaw's 2000 paper [3], in which she asked for role-specific education of future software engineers even during early baccalaureate degree programs, the SEE community has developed and published about approaches to teach not only theory (like formal methods [4] or architecture [5]), but also role-specific techniques (e.g., DevOps [6] or testing [7]), and of course soft-skills (e.g., stakeholder interaction [8] or teamwork [9]).

To teach this plethora of knowledge and skills software engineering (SE) students need after graduation, project-based learning (PBL) has emerged as the dominant pedagogical approach due to its multifaceted benefits [10]-[15]. Firstly, PBL provides students with a hands-on, practical experience that bridges the gap between theoretical knowledge and real-world

application [16]. In the context of software engineering, where problem-solving skills and practical implementation are paramount, PBL offers an effective platform for students to engage in authentic software development projects [17]. This experiential learning not only enhances their technical competencies but also fosters a deeper understanding of the software development life cycle, teamwork, and project management to deliver quality products within a specified time.

However, in focusing on all these benefits, the PBL course largely focuses on the software specification and construction process. While external [18] or simulated [19] stakeholders often serve to elicit requirements, validate design, and approve the final product, the technologies that make it into the software product are usually chosen ad-hoc. This means that students pick technologies (such as which library or framework to use) by familiarity, or random search. As a result, even the students who built a software product often have at best a superficial understanding of third-party components that are part of it, with limited or no understanding of cybersecurity risks, post-deployment maintenance requirements, and with a considerable amount of technical debt [20], [21] resulting from poor component choices.

A solution to this issue is to incorporate supply chain risk management (SCRM) into PBL approaches [22]. SCRM is the commonly accepted term to describe supplier-integrator relationships in contracted software development. SCRM education aims to teach students sensitivity about incorporating third-party software into their software products, thereby limiting the risk of malfunction and cyberattacks after deployment and improving ease of maintenance [23]. Cybersecurity vulnerabilities can appear in custom contracted software, open-source software, and even reused software from prior projects. Developers generally think about software that can be freely acquired, without regard to latent vulnerabilities, and students are no exception [20]. While SCRM curricula are freely available [24], their adoption has been slow [15], with only few approaches specifically focusing on SCRM at all [22]. Almost no PBL approaches exist that incorporate it. The repercussions in students' learning outcomes are threefold:

1. Students' justifications for why a component must be adopted are at best implicit, often due to emerging needs during implementation. In consequence, project stakeholders are unaware of which components (and possible vulnerabilities) are included in the product [23].

- Students do not evaluate advantages and disadvantages of an acquisition, e.g., by voting among alternatives. As a result, they often select the first choice, without considering usage rights [42].
- Students rarely change their mind about an acquisition, or only change their mind after much effort trying to get a component to work. This may result in technical debt throughout the project due to poorly managed risk assessment and problem identification [24]. This causes students to struggle through implementation issues until they achieve a passing, but perhaps inadequate solution (i.e., “hack” till it works).

In this paper, we aim to alleviate these issues by incorporating specific SCRM learning outcomes in a project-based software development course. In previous work [25], we have proposed Technology Acquisition Plans (TAPs) to facilitate SCRM in PBL. TAPs are documents, written by students alongside the specification and implementation effort throughout the project, to make explicit the choices in technology adoptions. TAPs are designed to generate sensitivity for the challenges above and serve as a proactive mechanism to reduce technical debt. Since proposing TAPs, we have implemented their use in an existing project-based software development course, and empirically evaluated their impact.

This paper is structured as follows. Section II discusses related work on project-based learning and supply chain risk management and briefly introduces the key principles of TAPs from previous work [25]. In Section III, we discuss how we introduced TAPs into an existing PBL course with particular emphasis on fostering repeatability. In Section IV, we describe our empirical method to evaluate the impact of TAPs. Findings are discussed in Section V. Finally, Section VI concludes this paper and outlines future work.

## II. BACKGROUND AND RELATED WORK

### A. Project-Based Learning in SEE

Project-based learning is an instructional approach, in which the learner is asked to produce artifacts that meet some learning objective in a self-directed fashion by making use of relevant course material. This can take many forms, such as “microprojects” that accompany relatively small instructional units, usually on very technical topics [1], [12], to semester-long course assignments with several milestones [11], [13], [27], [28]. Multi-semester studio-style projects [29], [30], or (as in our case) course-encompassing semester-long collaborative projects, much akin to capstone courses [31] have also been proposed. The key advantage in project-based learning is that it focuses on knowledge discovery (as opposed to rote memorization [32]) and fosters student enthusiasm as well as creativity [33], [34]. However, project-based learning often does not convey difficult theoretical relationships [35], like those pertaining to SCRM [22]. Most recently, SEE research has attempted to add theory-heavy instruction by increasing realism in project-based courses. Aiming at providing learners with experiences to increase their industry-readiness, such approaches make use of realistic case examples [11], [17], [35]–[37], or aim to put theory into practice by involving real [8], [18],

[38], [39] or realistic stakeholders [19], e.g., through role-playing.

### B. Supply Chain Risk Management and Software Acquisition

Supply Chain Risk Management (SCRM) describes activities carried out throughout the software development project to assess, monitor, and mitigate possible risk to the supply chain in software development [20]. *Risk* in this sense refers to the threats to project success due to the adoption of off-the-shelf components. This may include, but is not limited to inadequate staffing (i.e., insufficient knowledge to make adequate use of the component, [21]), introduction of security vulnerabilities [22], [40], or inadequate justification, monitoring, or communication of adoption choices to stakeholders [20], [22]. Shoemaker and Kohnke have proposed a Software Acquisition curriculum [24] that is freely available. The SCRM curriculum provides the following 12 learning outcomes (LOs):

- Provide a justification for the need to acquire a product
- Define a comprehensive testing and assurance plan
- Implement and publish an effective Request for Proposals
- Create a secure specification of requirements
- Execute a Technology Acquisition bidding process
- Make enforceable contracts incl. liquidated damages
- Perform monitoring and control on the build process
- Devise and execute an effective audit and control process
- Execute an effective Problem Resolution Process
- Ensure follow-up on action items has been completed
- Inspect and accept products with assurance
- Ensure the correctness and security of reusable code

A ready-to-use course design consisting of reading and lecture material, supplier and acquirer checklists, and surveys is available with the curriculum [24].

### C. Technology Acquisition Plans

We proposed Technology Acquisition Plans in [25] to integrate SCRM learning outcomes in existing PBL courses in baccalaureate degree programs in which no dedicated SCRM course exists, to specifically address the repercussions from Section I. TAPs are documents<sup>1</sup> written by students involved with technical aspects of the software development project before a third-party component is officially adopted by the team. The TAP serves as a proposal made to the rest of the team. Its purpose is to:

- formulate a use case for acquisition of a technology, including its disadvantages;
- formulate risks to acquisition success (i.e., what challenges arise when using the technology) and threats to success;
- list associated product requirements (if available) that are met by this technology; and
- list the acquisition timeline (i.e., how long from acquisition to effective application in the project).

The Technology Acquisition Plan highlights copyright issues and documents risks and alternatives for the project stakeholders, is formally voted on for approval by the entire course cohort, and the adoption process is then monitored in a

---

<sup>1</sup> A free-to-use template is available at [26].

way such that problems can be dealt with before they increase technical debt of the project. In consequence, TAPs specifically focus on LOs 1, 5, and 9 listed above. The remaining SCRM learning outcomes listed above are either typical for construction-centric PBL courses (LOs 2, 4, 7-8, 10-12) or are beyond their scope (i.e., financial and contract matters in LOs 3 and 6, respectively).

In the next section, we outline how TAPs were integrated.

### III. INTEGRATING TAP IN A SOFTWARE DEVELOPMENT COURSE

TAPs have been implemented using the method outlined in [25] into a semester-long project-based software development course taught at the State University of New York at Oswego. In the following, we shall outline course design, team responsibilities, and a semester plan showing aspects that were newly introduced to integrate TAPs and SCRM considerations.

#### A. Course Design

The course is similar to capstone project courses [41]. However rather than emphasizing individual or small-group projects, it requires all 20-30 students enrolled in the course to work together in task-specific teams on a large-scale software product. This includes eliciting and documenting requirements for a prescribed software product (the project is chosen by the instructor in collaboration with external stakeholders), implementation, and quality assurance. In this sense, the course relies heavily on knowledge from previous courses. Since the aim of the course is to “bring together” what students have learned in their degree programs until that point, there is little lecturing. Instead, all class meetings are dedicated towards sprint-planning, progress report updates, and groupwork, with most specification, implementation, and quality assurance taking place between class meetings. Nevertheless, depending on project progress, the instructor typically delivers ad-hoc mini-lectures on various topics, including requirements modeling and prioritization, milestone planning, criticality analysis, and empirical methods (including statistics).

TABLE I. LEARNING OBJECTIVES (LOS) AS FILED FOR ACCREDITATION

Learning Outcome Description		Mapped SCRM LO
1	Demonstrate effective collaboration on a team project.	-
1'	<b>Justification of the need to acquire a product</b>	1
2	Demonstrate understanding of the software development process.	2, 10
3	Create, work in, manage, and communicate within a team as well as between teams.	-
4	Apply software metrics.	7
5	Use current tools and methods to plan, analyze, design, test, measure, and manage software projects.	8
5'	<b>Execute a secure bidding process including a financial impact analysis.</b>	5
6	Plan and conduct static and dynamic software artifact quality assurance.	11
7	Articulate the advantages and disadvantages of current software life cycle models	12
8	Articulate advantages and disadvantages of design choices and solution alternatives.	4
9	Communicate impact of solution alternatives to product success	2, 10
9'	<b>Create and execute an effective Problem Resolution Process</b>	9

The learning objectives (LOs) are shown in Table I. SCRM-specific learning outcomes are mapped in the right column, referencing the LOs from Section II.B, where applicable. Newly added LOs to satisfy SCRM as discussed in Section II.C are highlighted in bold.

#### B. Team Responsibilities

Students self-assign to exactly one of the following teams. The instructor provides guidance, based on students' experiences, interests, and team-strength. Teams are tasked with producing artifacts pertaining to their areas of expertise. The teams are as follows:

- *Requirements*, in which students previously or concurrently enrolled in a requirements engineering course produce an IEEE29148-compliant specification that governs implementation efforts.
- *QA*, in which students previously or concurrently enrolled in a quality assurance course are responsible for independent validation and verification of all project artifacts, select quality criteria and metrics, ensure testability, and conduct unit testing to produce the quality assurance report. This team may also be in charge of validating deployment instructions or other documents.
- *Usability*, in which cross-enrolled graduate students develop a user interface prototype, conduct heuristic analyses of its usability, plan and execute empirical usability tests of the final product with external participants, and draft a usability assessment report.

These teams' responsibilities mostly include non-programming tasks. By contrast, the following three teams' responsibilities are more implementation-centric. These are:

- *DB / Networking / Security*, in which students previously or currently enrolled in database management, networking, or software security courses develop a database and suitable endpoints, write deployment and integration plans, and conduct a penetration test at the mid-point, and end of the project, resulting in a penetration test report.
- *GUI*, in which students who self-identify as having user interface experience or express their strong desire, select appropriate technologies and interface with *Usability* and *Engine* teams to implement the prototype.
- *Engine*, which typically comprises all remaining undergraduate students and is usually the largest team. The engine team is the main driver of implementation, as students interact with all other teams to implement the core functionality of the product, offering endpoints to *GUI* and consuming those provided by *DB*.

Team composition may dynamically change, or students may be reallocated, if the development situation necessitates such a change, or if work tickets run out for a team. For example, the *Requirements* team usually has a high-workload in the beginning of the semester, while *QA* takes place during the mid-point and final semester stretches. Times of low workload for one team usually coincide with peak workload for other team, hence necessitating allocation of additional students to different teams to balance the workload.

A leader is selected for each team (or typically emerges from the team composition) to connect with other teams and facilitate progress reporting. Throughout the course, the instructor gives advice and helps students with progress planning and acts as product owner (to assist in engineering choices), project manager (to overcome institutional challenges like access to a department server), stakeholder (depending on whether external stakeholders are participating in the specific semester), and sometimes facilitator (when interpersonal challenges arise). Instructor responsibilities sometime involve role-playing [19].

Common course tools (e.g., Github for artifact management, Discord for asynchronous communication) and the project topic are typically selected by the instructor. In all other aspects (e.g., which language or libraries to use), selections are made by the students. This is notably the case for the selection and adoption of third-party off-the-shelf components and frameworks. Popular choices include ReactJS, MongoDB, OpenLiberty, Java Web Tokens, and Google OAuth. However, depending on the project, libraries to be used in conjunction with these technologies can vary quite a bit and may be spontaneously adopted by a student during implementation, without proper consideration of risks and possible challenges.

### C. TAP and SCRM Implementation Strategy

Improper consideration and ad-hoc adoption of third-party off-the-shelf components has traditionally resulted in a lack of reporting of the composition of the software product and can introduce SCRM risks. In other words, although students are encouraged to maintain communication with the stakeholder and defend their engineering choices, they rarely do so as a natural part of their work routine and struggle to provide an in-depth justification of their selection. Consequently, stakeholders are left with incomplete knowledge about the components in the product [22]. Moreover, security risks are not usually uncovered until the DB/Networking/Security team conducts penetration tests [23], resulting in a reactive mode (requiring fixes to the codebase) instead of proactive (designing to minimize threats). In addition, a recurring challenge for the instructor is to impart a sensitivity about usage rights to students (i.e., only because a library is available does not imply that it is permitted to be used, see also [42]).

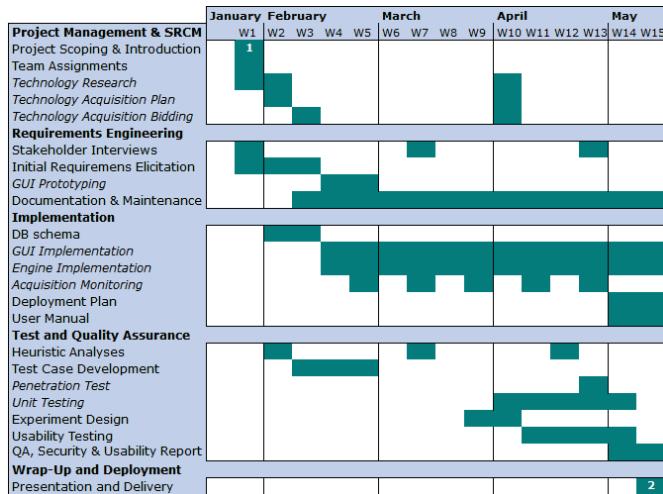


Fig. 1. Adapted Semester Plan with TAP integration to meet SCRM LOs.

To overcome these challenges, we adopted the strategy proposed in [25]. Fig. 1 shows the adapted course schedule including the use of TAPs to meet SCRM learning outcomes in the context of typical team activities throughout the semester. As outlined in Section II.C, TAPs are documents written by implementation-centric teams, i.e., the students who are the most impacted by adoption and repercussions. As can be seen, TAPs mainly impact two development phases. Activities in the phases impacted by TAP are shown in *italics*. Acquisition-related activities that are new to the SCRM effort for this course include:

**Project Management & SCRM.** In this phase, the project is scoped, and team assignments are made. Students seek to understand the project vision and the instructor facilitates a shared understanding of the project goals. While the teams are preparing the requirements elicitation phase, experience shows that implementation-centric teams are either idle or begin by making some implicit choices frameworks and libraries, without informing those in charge of documentation about the implications. Yet, these implications can impact the requirements engineering effort. Therefore, implementation-centric teams are instructed to delay concrete technical work in favor of making some high-level conceptual suggestions about which technologies they seek to adopt after thoroughly researching advantages, disadvantages, and alternatives. For example, Golang and Python are both suited to develop web applications, however which of the two is the better choice depends on factors such as nature of the project and available talent in the team. One TAP is filled out for each technology adoption. All TAPs are discussed in class with all teams, weighing alternatives and ratifying choices in a bidding process. A majority vote replaces the financial bids from [24].

The result of this is a set of TAP documents using the template provided in Section II.C, one for each acquisition candidate component, along with a timeline for adoption, possible risks, and possible countermeasures against risks. This effort front-loads technology choices that are ordinarily made during implementation. While not all possible choices can be adequately foreseen, a second TAP and bidding process is conducted around week 10 of the project in an effort to adopt additional technologies in a systematic fashion, mostly for testing and quality assurance.

**Implementation.** In this phase, the bulk of the project is implemented by the *Engine*, *GUI*, and *DB/Networking/Security* teams. During this phase the *Requirements* team takes on the burden of monitoring the acquisition processes regarding whether their acquisition goals are met, if the acquisition timelines are maintained or overrun, and if any risks emerge to threaten project success. This directly impacts GUI and engine implementation the most, as it requires them not just to share progress with the *Requirements* team in a high-level fashion, but also share implementation details. The result is that all specification-centric teams are kept in the loop about implementation details from which they would normally be excluded, as these details are now captured in the requirements specification. As acquired software continues to be integrated, all teams are also put in a position to identify risks to project success early and replan the development effort.

**Impact on other Activities.** Other activities impacted by SCRM concerns are GUI prototyping, unit and penetration testing. In contrast to GUI and engine implementation, these phases are only slightly affected in the sense that the choices of which libraries and frameworks to use for these activities depends on the adopted components during implementation.

In the next section, we discuss how we evaluated the impact of using TAPs on students' self-reported SCRM knowledge.

#### IV. APPROACH TO EVALUATE TAP APPLICATION

To assess the impact of TAPs on SCRM learning objectives, we conducted a mixed-subjects experiment, consisting of between-subject and within-subject measures. In this section, we outline the experimental design, the pedagogical intervention as well as the instrument we used to gather data and the method of analysis. We then discuss results on students' self-reported SCRM knowledge from the PBL course in Section V.

##### A. Study Design and Pedagogical Intervention Procedure

As outlined in Section III, the PBL course runs every semester. Assessment in this course takes the form of evaluating individual students' performance throughout the semester-long project, timeliness, and quality of achieving self-assigned milestones, performance in class presentations, collaboration on team design, and development progress. As tools for assessment, we use peer evaluations. These are captured at the midpoint and end of semester and ask students to rate themselves and their immediate team members in terms of team meeting performance, contribution of conceptual work, and amount and quality of work produced on a 5-point scale. Instructor assessments use the same method. In this study, we aimed to assess the impact of TAPs on SCRM-related learning outcomes, specifically LO1, LO5, and LO9 from Section II.B. To do so, we used the survey instrument discussed in Section IV.B as shown in Fig. 2.

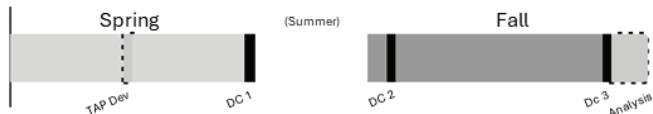


Fig. 2. Adapted Semester Plan with TAP integration to meet SCRM LOs.

The timeline of this study implemented our proposed assessment plan in [25] and is as follows. In the Spring semester, the PBL course was conducted as usual, i.e., without using TAPs and without SCRM learning outcomes. Roughly around the midpoint of the semester, SCRM research was conducted and the curriculum in [24] was adapted to suit the needs of this course (TAP Dev in Fig. 2) to minimize the shortcomings in the course identified in previous work (see Sections I and III.C). TAPs as a solution to introduce SCRM LOs were developed and the survey instrument to be used at the end of the semester was developed. The semester continued normally. At its end, and after ethics approval was granted by the institution's review board, the survey was administered as part of the end-of-semester assessment outlined above. This constitutes data collection point DC1 in Fig. 2 and the between-subjects pre-treatment component. Students were informed that participation was voluntary and neither resulted in extra credit nor penalty.

Eighteen of the 27 students enrolled in the course participated, two were graduate students.

After the summer break, another delivery of the course took place in the Fall. This time, Technology Acquisition Plans as outlined in Section II.C were introduced in the first week of class, and the template in the link in Footnote 1 was made available to students. Students involved with implementation tasks were asked to ascertain which technologies they most likely require to implement the vision while other teams were conducting requirements engineering and interviews. The semester then proceeded as usual, following the schedule shown in Fig. 1. At the beginning of the semester as well as at the end, the same survey instrument used in DC1 was administered to the students in the Fall, again on a voluntary basis. Of the 26 students enrolled in the course, four of which were graduate students, 19 participated in the within-subjects pre-treatment condition (DC2 in Fig. 2) and 20 students participated in the within-subjects post-treatment condition (DC3 in Fig. 2, only one graduate student).

##### B. Instrument & Analysis

For the three data collection points DC1-3, we adopted the SCRM capability maturity assessment checklist provided as part of the Software Acquisition curriculum [24]. Since we adopted a subset of LOs, we only adopted relevant checklist items, i.e., LOs 1, 5, and 9 to include in our instrument. The instrument is available in Appendix A. We adopted the same scale as suggested in [24], i.e., a 5-point semantic differential consisting of the following items, definitions, and point values:

0. **Not Done:** The student selects this answer option if they feel that the activity suggested in the checklist item was not done at all by themselves or anyone else in class.
1. **Performed:** The student selects this answer option to indicate the activity suggested in the checklist item was perhaps done by someone else in class in an ad-hoc manner (i.e., they "winged it").
2. **Managed:** Indicates that the activity was planned for ahead of time, assigned to a specific person, and its execution was tracked (e.g., by talking about it regularly).
3. **Predictable:** Indicates that detailed measures of performance were collected, and results were operationalized (e.g., writing down decisions or voting).
4. **Optimized:** Indicates that continuous improvement is enforced by quantitative data that is obtained from the execution of the defined processes as well as from piloting innovative ideas and technologies (e.g., by using "toy examples" or "prototypes").

Data was collected using Google Forms and non-responses (e.g., answering in patterns, discontinue participation, or giving nonsense answers) were pruned from the data set. The remaining responses were analyzed by summing up the score for each survey item, using the point values in the list above. This score was divided by the total possible score from all survey participants (i.e., the maximum sum if all participants answered "optimized" for the survey item). This yields a measure which we call "course agreement" and indicates all participating students' self-reported, subjective level to which degree SCRM-related activities were performed. For each learning outcome, we calculate the agreement percentage by summing up the

course agreement pertaining to each LO. In the following, we call that the “LO agreement.” We hypothesize that the LO agreement is highest in data from DC3.

However, as we conducted data collection in DC1, we noticed that students in that semester responded by giving unexpectedly high scores for most survey items. It must be noted that these students received no dedicated SCRM training and were “surprised” by SCRM-related questions at the end of the semester survey. Since data collection took place as part of peer evaluations and assessment for final grades, it is possible that students answered with high scores, conflating voluntary study participation with grade assessment. This may have resulted in a ceiling effect and is a threat to validity. To mitigate this threat, we define our hypotheses  $H_x$  for all learning outcomes ( $x = \{1, 5, 9\}$ ) as follows:

- $H_x$ : The LO agreement for  $LO_x$  is significantly higher in DC1 and DC3 than in DC2 (i.e.,  $LO_{1\wedge 3} > LO_2$ ).

We conducted T-Tests to test for significance. We assumed normal distribution of results and validated this assumption with pre-hoc F-Tests to determine T-Test subtype. In case F-Tests yield no normally distributed data for DC1 but does yield normal distributions for DC2 and 3, we take this as evidence that a ceiling effect is present in the data. In that case, we will exclude DC1 from hypothesis testing and subsequent conclusions but will report it for complete disclosure in Section V. We interpret significant T-Test results as evidence in favor of our hypotheses.

## V. EVALUATION RESULTS

In this section, we discuss the findings of our survey regarding each SCRM LO. For each LO, we discuss the course agreement and check for significant differences between the data collection points. We supplement the data with classroom observations. Raw data and analyses are available in the link in Footnote 1. We conclude this section by contextualizing findings with our classroom experience.

### A. Learning Outcome 1: Justifying Acquisitions

Learning Outcome 1 states that a course shall provide students with the ability to “perform a justification and need to acquire a product.” This LO therefore directly addresses students’ typical tendency to adopt technologies such as libraries and frameworks in an ad-hoc fashion. TAPs specifically ask students to find a rationale for adoption as well as possible alternatives. The degree to which the introduction of Technology Acquisition Plans helped can be seen in Fig. 3.

For all data collection (DC) points, Fig. 3 shows the course agreement for answer options, i.e., the percentage of students who answered “not done”, “performed,” etc. for this LO. It also shows the LO agreement, i.e., the students’ self-reported degree to which they meet the learning outcome (calculated by summing up the course agreement score and dividing by total number of responses). As can be seen, most students answered that proper justifications were “managed” in all three DCs. Curiously, even those students participating in DC1, who had no SCRM training, answered rather highly, with about 45% of students categorizing their SCRM effort as predictable or

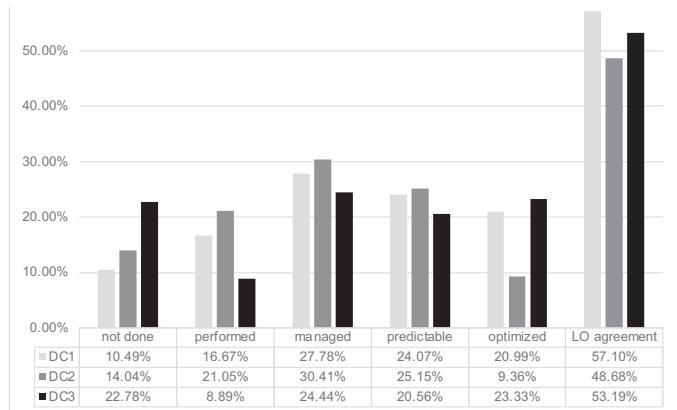


Fig. 3. Course Agreement and LO Agreement for Learning Outcome 1.

optimized. This suggests that despite not having been instructed about the need for justifications, students naturally justify technology adoptions. This contrasts with our classroom impressions that we will discuss in Section 0. For DC2 and DC3, a clear positive trend is visible: students in this semester received SCRM training and were asked to justify the need to adopt each technology in dedicated TAPs. As the semester continued, the number of adoptions increased and hence, the number of justifications increased also, thereby exposing students to more activities pertaining to this LO. The data supports shows that in consequence, the agreement for LO1 is higher in DC3 than in DC2, with DC1 being the highest. Table II shows the F-Test and T-Test results to check significance.

TABLE II. LO AGREEMENT COMPARISONS FOR LEARNING OUTCOME 1

Descriptives	DC1	DC2	DC3
Mean	57.10%	48.68%	53.19%
Variance	6.55%	5.04%	10.94%
Sample Size	18	19	20
Student’s T	DC1 vs DC2	DC2 vs DC3	DC1 vs DC3
dF	34	34	35
F-Test	0.293	0.056	0.151
p	0.148	0.309	0.342

F-Test results in Table II indicate no normal distributions of the variances for any DC. This indicates that a ceiling effect for DC1 is not present in the data. Furthermore, T-Test results show that no difference in means is significant. We therefore must reject hypothesis  $H_1$ : using TAPs did not significantly improve students’ self-reported proficiency in justifying the need for technology adoptions. Nevertheless, a tendential positive trend can be seen: as students complete the project, their exposure to secure acquisition activities and justifications is increasing (see DC2 vs. DC3 in Fig. 3), and as we will outline in Section 0, happens in a more structured fashion when using TAPs as than without using TAPs (DC1 vs. DC3).

### B. Learning Outcome 5: Bidding Process

Learning Outcome 5 states that any technology acquisition shall “execute a technology acquisition bidding process.” In previous iterations of the course, this was not part of the typical

project activities, but has been introduced together with SCRM training and TAPs as part of the software documentation, as can be seen in Fig. 4. Like Fig. 3 before, course agreement to survey items pertaining to LO5 and the LO agreement are plotted as percentages of all students' responses.

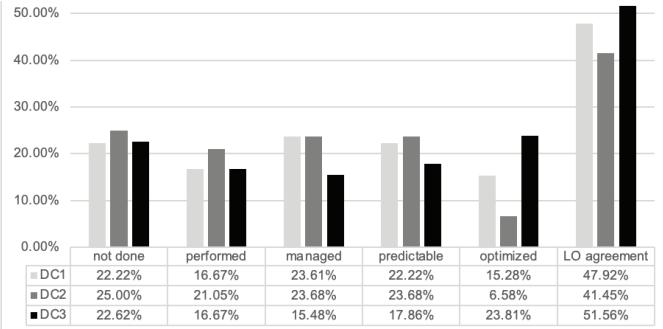


Fig. 4. Course Agreement and LO Agreement for Learning Outcome 5.

As can be seen, course agreement is higher for DC2 and DC3 for each answer option. The exception is “optimized” for DC2, and “managed” and “predictable” for DC3, which are lower than those for DC1. This data suggests that using TAPs and repeatedly voting on the adoption of technology made students more aware of the need to adopt technology in a systematic fashion. The LO agreement seems to suggest the same, with DC3 scoring the highest. This indicates that students without SCRM training may be conflating “bidding” on a software acquisition with “selecting” a component without consideration of alternatives. TAPs seem to have the effect of resolving this convolution, as is shown by the LO agreement being highest for DC3 by a 10 percent-point margin (cf. DC2). However, as can be seen in Table III, this effect was not significant at the  $p < 0.05$  level. Rejecting the assumption of normally distributed variances using F-Tests shows no ceiling effect in DC1. T-Test results show that the difference between DC3 and DC2 as well as DC1, respectively are not significant.

TABLE III. LO AGREEMENT COMPARISONS FOR LEARNING OUTCOME 5

Descriptives	DC1	DC2	DC3
Mean	47.92%	41.45%	51.56%
Variance	8.87%	6.30%	11.38%
Sample Size	18	19	20
Student's T	DC1 vs DC2	DC2 vs DC3	DC1 vs DC3
dF	33	35	36
F-Test	0.239	0.108	0.305
P	0.241	0.147	0.363

We must therefore reject hypothesis H<sub>5</sub> and assume that while TAPs and voting on their approval has a positive effect on students' self-reported exposure to secure bidding processes, this effect was marginal.

#### C. Learning Outcome 9: Problem Resolution

Finally, Learning Outcome 9 requires students to “create and execute a problem resolution process.” A specific component of TAPs is to identify risks and a timeline for successful adoption,

along with a strategy to identify and mitigation problems. In doing so, it clearly exposes students to proactive risk management during the software acquisition process.

Fig. 5 shows that this is successful. As can be seen from the figure, most students (ca. 45%) at the end of the project (DC3) agree that problem identification and resolution activities were predictably executed or executed in an optimized fashion. By contrast, most students in the same semester (DC2) said such processes were “managed” and problems were “predictable” (53%). Both of these scores are higher than in students without SCRM training (DC1). This confirms the positive impact of TAPs, whose purpose is to make explicit software acquisition needs and possible risks and front-load the adoption process management to the beginning of the semester. Hence, at DC2, students may have answered in the positive under the assumption that TAPs would enable to “predict” and “optimize” problem resolution. What is surprising, however, is that some 17% of students at DC3 compared to only 9% of students at DC2 indicated that problem resolution was not done.

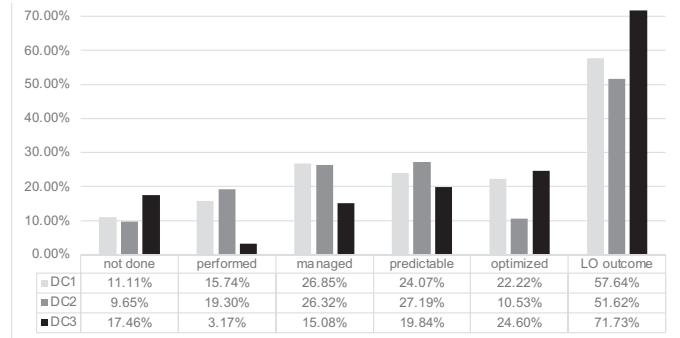


Fig. 5. Course Agreement and LO Agreement for Learning Outcome 9.

We have no explanation for why this number increased, other than this being an artifact of participation attrition. The number of students discontinuing the survey at its end (and hence being removed from analysis) increased. This can be seen in Table IV.

TABLE IV. LO AGREEMENT COMPARISONS FOR LEARNING OUTCOME 9

Descriptives	DC1	DC2	DC3
Mean	57.64%	51.62%	71.73%
Variance	6.64%	5.05%	7.14%
Sample Size	18	18	14
Student's T	DC1 vs DC2	DC2 vs DC3	DC1 vs DC3
dF	33	25	28
F-Test	0.289	0.248	0.437
P	0.230	<b>0.016 (sig.)</b>	0.072

Table IV lists a sample size of 14 for DC3, which is 6 students less than at DC3 in Table III, compared to one student for DC2. Again, as for LO1 and 5, we rejected the assumption of distribution normality for all DCs using F-Tests and conducted T-Tests. The results are also shown in Table IV and indicate that the LO agreement was dramatically higher for DC3 than DC2 and DC1, respectively; with no significant probability

of a possible ceiling effect in data, despite attrition. The within-subjects difference is significant at the  $p < 0.05$  level. This means that we can accept these results in favor of our hypothesis  $H_0$ : Using TAPs to structure and guide risks to the software adoption process has a significantly positive effect on students' self-reported ability to predict and optimally recover from problems that arise during their adoption.

#### D. Interpretation of Findings in Light of Classroom Observations

In summary, the results in Sections V.A-0 show that using Technology Acquisition Plans as a means to foster Supply Chain Risk Management learning outcomes has a positive effect on students self-reported proficiency, but not unilaterally so. There seems to be only a marginal effect on their self-reported ability to justify the need for acquisition (LO1), a positive, but not significant effect on students' engagement when bidding on acquisitions (LO5), and a strikingly positive effect on their ability to plan for and systematically resolve problems during the adoption process (LO9). In the following, we interpret these findings in light of classroom observations.

**Learning Outcome 1: Justifying Acquisition.** It was surprising to us to see that students did not self-report higher proficiency in justifying software acquisitions using TAPs, neither compared to students who did not use TAPs (DC1), nor compared to themselves after having applied TAPs throughout the entire project (DC3). There might be two reasons for this finding. On one hand, what we initially considered a ceiling effect in the data (see Section IV.B) may in fact have been an adequate post-project assessment of SCRM activities. We initially believed high scores could be caused because students conflated study participation with grade assessment. However, upon closer inspection of the findings, it is possible that the very nature of the survey, which asked about LOs and activities pertaining to SCRM, may have caused students to reflect on their SCRM-related activities even without having been formally instructed therein. This is an example of instrumentation bias and an additional threat to validity. Nevertheless, the results show that students even without SCRM knowledge do indeed justify the need to adopt a component, library, or framework, however, do so more systematically with the use of TAPs. This is in line with our classroom observations, which show that students identify the need to adopt a component, however they do not write this decision down, nor do they look for alternatives or think critically about the adoption process. TAPs have clearly changed this for the better, making the adoption process smoother in almost all cases and in some cases identify preferable alternatives to inform the bidding process (LO5). Without TAPs, only a single student would decide on an acquisition. By contrast, creating TAPs usually involved several students, leading to much better information flow from implementation-centric to specification-centric teams, and for the first time since offering this PBL course, caused explicit content descriptions of the included third-party components to be part of the software documentation.

**Learning Outcome 5: Bidding Process.** There was a positive effect of TAPs and the bidding process TAPs entail that became apparent in Section V.B. However, this effect was marginal. A possible reason for this is that in our experience, the

front-loading of software acquisition planning to the beginning stages of the project (DC2) did not work effectively. Frequent TAPs had to be drafted almost throughout the entire semester. The "bidding process" took the form of issuing a majority vote by the entire course cohort to select the best among all identified alternative components. This process was taken seriously at the very beginning of the course with many discussions and not always unanimous decisions. But roughly at the mid-point of the semester (between DC2 and DC3), as implementation continued, technology adoptions happened frequently, with votes on adoptions happened almost weekly. As a result, the bidding process became tedious, repetitive, and students stopped seeing it as important. It was noticeable how students who felt that a specific technology adoption would not impact them, would "tune out" of the bidding process and supply blanket consent, leading to frequent unanimous decisions and less critical discussions about adoptions. For example, team *Engine* paid noticeably less attention to team *QA*'s endeavor to adopt Selenium for front-end unit testing.

**Learning Outcome 9: Problem Resolution.** While front-loading the acquisition planning did not work as the instructors expected, the attempt to plan the acquisition of third-party components had the significantly positive effect of forcing students to commit to a timeline of when adoption will be complete, force them to think of alternatives, and carefully monitor the adoption process. This caused students to work on integrating components much more stringently and re-assigning tasks dynamically to teams and persons when problems occurred, even across team boundaries. While this did not remove all unforeseen challenges, it reduced them to a dramatic degree compared to previous semesters. In the aforementioned example between teams *Engine* and *QA*, there were frequent instances where GUI prototyping, unit testing, and penetration testing were simplified because the students could plan their activities much earlier, knowing which technologies would be involved and what the code base looks like. The handoff between implementation, testing, and deployment was improved and free of the typical delays from previous semesters. These positive impacts are highlighted by the data from Section 0

## VI. CONCLUSION AND FUTURE WORK

In this paper, we have presented Technology Acceptance Plans to instruct Software Supply Chain Risk Management (SCRM) learning outcomes in a project-based software development course. In the course, all students work in parallel on the same software product, from specification to implementation, quality assurance, and delivery to external stakeholders. To alleviate issues pertaining to SCRM, specifically increasing students' ability to justify the need for third-party software acquisition, to engage in "secure bidding", i.e., carefully considering alternatives, and to systematically plan for problem resolution and monitor risks. To measure the impact of Technology Acceptance Plans, we adopted the checklist provided with a freely available Software Acquisition curriculum [24] alongside the learning outcomes and activities proposed therein. We measured the students' self-reported level of proficiency when engaging in these activities. Results show that while their self-reported ability to justify adoptions did not improve (LO1), classroom behavior indicates that information flow about adopted components and their justifications became

more explicit. While a regular bidding process was considered tedious, front-loading the acquisition plans and careful deliberation that comes within bidding (LO5) marginally positively influenced students' ability to plan the work that depended on the acquisition. Finally, strongly significant positive results regarding students' ability to monitor and act upon challenges were identified, both by our survey as well as classroom observations (LO9).

As with any study, threats to validity remain despite careful design of the empirical evaluation. Other than the possible ceiling effect in the data mentioned in Section IV.B, which we were able to rule out, three threats to validity remain. Firstly, it must be noted that the instrument we used to assess SCRM learning outcomes was not made by us, but by the same authors of the SCRM curriculum in [24]. Moreover, the instrument was designed as a checklist, not as a measure to assess learning outcomes, giving rise to a threat to construct validity. However, we believe that an instrument supplied with the curriculum is more fitting and more objective than a newly created instrument by us, thereby minimizing the threat to validity. Secondly, conclusion validity may be impaired due to fact that we used the same instrument, which asks for activities that have been done in the project, both at the beginning and the end of the project (in the Fall, DC2 and DC3 in Fig. 2). Since most project activities had not been completed yet, this could have skewed data and thereby may have impaired results. We mitigated this threat by delaying DC2 until some project activities have concluded instead of at the onset of the semester, which would have inquired about previous project experience instead uniformly of the current project experience. Finally, conclusion validity is impaired by the chosen statistical measure (i.e., T-Tests), which are sensitive to small sample sizes. However, in all DCs, sample size was around the recommended minimum for parametric tests [43] and data did not include extreme outliers, hence giving credence to our findings.

Future work shall investigate students' ability to justify the need for third-party software acquisition. Despite the TAP template explicitly requiring students to articulate a proper justification, results show that TAPs did not improve their self-reported ability to do so (LO1). We conjecture that this may be due to the fact that students conflate project-specific reasons (e.g., "without component X, we cannot complete task Y") with personal ones (e.g., "it makes my job easier, so therefore we need it"). A dedicated experiment investigating students' ability to make project justifications including acquisition-related ones will shed light onto this question. A reasonable avenue is to apply SCRM-related metrics as proposed in [23]. Moreover, an open question remains how to introduce SCRM aspects in PBL courses in earlier semesters than the capstone-like course selected for this investigation.

## REFERENCES

- [1] B. Malik and S. Zafar, "A systematic mapping study on software engineering education," *Int. Journal of Educational and Pedagogical Sciences*, vol. 6, no. 11, pp. 3343 – 3353, 2012.
- [2] V. Garousi, G. Giray, E. Tuzun, C. Catal, and M. Felderer, "Closing the Gap between Software Engineering Education and Industrial Needs," in *IEEE Software* 37(2), 2020, pp. 68-77.
- [3] M. Shaw, "Software Engineering Education: A Roadmap." In Proc. Future of Soft Eng, 2000, pp. 371-380.
- [4] A. Kelley Sobel, "Experience Integrating a Formal Method into a Software Engineering Course," *ACM SIGCSE Bulletin* 28(1), 1996, pp. 271-274.
- [5] C. R. Rupakheti, S. V. Chenoweth, "Teaching Software Architecture to Undergraduate Students: An Experience Report," Proc. 37<sup>th</sup> IEEE/ACM Int. Conf. on Software Engineering, doi: 10.1109/ICSE.2015.177, 2015.
- [6] Ch. Jones, "A proposal of Integrating DevOps into Software Engineering Curricula," Proc. Software Engineering Aspects of Continuous Development and New Paradigms in Software Production and Deployment, 2018, pp. 33-47.
- [7] V. Garousi, A. Rainer, P. Lauvås Jr., A. Acruri, "Software-testing education: A systematic literature mapping," *Journal of Systems and Software* 165, doi: 10.1016/j.jss.2020.110570, 2020.
- [8] B. Brügge, S. Krusche, and L. Alperowitz, "Software Engineering Project Courses with Industrial Clients", *Trans. Comput. Educ.* 15(4), 2015, pp. 17:1–17:31.
- [9] Y. Tachikawa and T. Nakamura, "Education for requirements elicitation using group-work and role-play," in 2017 IEEE Global Engineering Education Conf. (EDUCON), 2017, pp. 780–783.
- [10] S. Karaçalli, and F. Korur, "The Effects of Project-Based Learning on Students' Academic Achievement, Attitude, and Retention of Knowledge: The Subject of 'Electricity in Our Lives,'" *School Science and Mathematics* 114(5), 2014, pp. 224-235.
- [11] B. Tenbergen, and M. Daun, "Industry Projects in Requirements Engineering Education: Application in a University Course in the US and Comparison with Germany," Proc. Hawaii International Conference on System Sciences, Invited Track on Software Engineering Education & Training, 2019.
- [12] E. F. Barry, "Micro Projects: Putting Light and Magic into Learning Computer Systems Concepts. In Proc. of the 2017 ACM SIGSCE Technical Symposium on Computer Science Education, 2017, p. 733.
- [13] S. Yadav, and J. Xiahou, "Integrated project based learning in software engineering education," Proc. of the International Conference on Educational and Network Technology, 2010.
- [14] M. Souza, R. Moreira, and E. Figueiredo, "Students Perception on the use of Project-Based Learning in Software Engineering Education," in Proc. 33<sup>rd</sup> Brazilian Symposium on Software Engineering, 2019, pp. 537-546.
- [15] M. Daun, A. M. Grubb, B. Tenbergen, "A Survey of Instructional Approaches in the Requirements Engineering Education Literature," Proc. 29<sup>th</sup> IEEE Int Conf Requirements Engineering, 2021.
- [16] S. Aldabbas, "Project-Based Learning: Implementation & Challenges," *International Journal of Education, Learning and Development* 6(3), 2018, pp. 71-79.
- [17] M. Daun, A. Salmon, Th. Weyer, K. Pohl, and B. Tenbergen, "Project-Based Learning with Examples from Industry in University Courses – An Experience Report from an Undergraduate Requirements Engineering Course," Proc. IEEE Conf. Soft. Eng. Education & Training, 2016.
- [18] B. Penzenstadler, M. Mahaux, P. Heymans, "University meets industry: Calling in real stakeholders", Proc. 26th IEEE Conf. Soft. Eng. Education & Training, 2013, pp. 1-10.
- [19] D. Zowghi, S. Paryani, "Teaching requirements engineering through role playing: lessons learnt", Proceedings. 11th IEEE International Requirements Engineering Conference, 2003, pp. 233–241.
- [20] D. Shoemaker, N. Mead, and A. Kohnke, "Teaching Secure Acquisition in Higher Education," *IEEE Security & Privacy* 18(4), 2020, pp. 60-66.
- [21] M. Keil, P. Cule, K. Lyytinen, and R. Schmidt, "A Framework for identifying Software Project Risks," *Communications of the ACM* 41(11), 1998, pp. 76-83.
- [22] N. Mead, A. Kohnke, and D. Shoemaker, "Secure Sourcing of COTS Products: A Critical Missing Element in Software Engineering Education," in Proc. 32<sup>nd</sup> IEEE International Conference on Software Engineering Education & Training, 2020, pp. 239-243.
- [23] N. Mead, C. Woody, S. Hissam, "The Measurement Challenges in Software Assurance and Supply Risk Management." White Paper, Software Engineering Institute. Carnegie Mellon University Digital Library, 2023. Accessed 24-01-2024. Available at: <https://insights.sei.cmu.edu/library/measurement-challenges-in-sw-assurance-and-scrm-white-paper/>

- [24] D. Shoemaker, A. Kohnke, and N. Mead, "Secure acquisition curriculum," Carnegie Mellon University, Software Engineering Institute, Digital Library, 2020. Educational Material, accessed 10/16/2020: <https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=637101>.
- [25] B. Tenbergen, N. Mead, "Adapting a Software Acquisition Curriculum to Instruct Supply Chain Risk Management in a Project-Based Software Development Course," Proc. 3rd Intl. Workshop on Software Engineering Education for the Next Generation (SEENG).
- [26] B. Tenbergen, N. Mead, "Technology Acquisition Plans to Foster Supply Chain Risk Management Learning Outcomes in Project-Based Software Development Courses: Raw Data and TAP Template". Zenodo, Jun. 06, 2024. doi: 10.5281/zenodo.11506417.
- [27] P. C. Blumenfeld *et al.*, "Motivating project-based learning: sustaining the doing, supporting the learning," in *Educational Psychologist*, vol. 26, 1991, pp. 369-398.
- [28] S. Boss and J. Krauss, *Reinventing project-based learning: Your field guide to real-world projects in the digital age*, 1st ed. Eugene, OR, 2008.
- [29] D. Garlan, D. Gluch, and J. Tomayko, "Agents of change: educating software engineering leaders," IEEE Computer 30(11), 1997, pp. 59-65.
- [30] T. Nurkkala, and S. Bradle, "Software Studio: Teaching Professional Software Engineering," In Proc. of the 42<sup>nd</sup> ACM SIGCSE Technical Symposium on Computer Science Education, 2011, pp. 153-158.
- [31] C.-Y. Chen, and P. P. Chong, "Software Engineering Education: A Study on Conducting Collaborative Senior Project Development," Journal of Systems and Software 84(3), 2011, pp. 479-491.
- [32] A. Sivan, R. Wong Leung, L. Gow, D. Kember, "Towards more active learning in hospitality studies", Intl. J. of Hospitality Mgmt 10(4), 1991.
- [33] F. Meawad, "The Virtual Agile Enterprise: Making the Most of a Software Engineering Course," in Proc. 24th IEEE Conf. Soft. Eng. Educ. & Training, 2011, pp. 324-332.
- [34] J. Burge, "Application and Appreciation: Changing Course Structure to Change Student Attitudes," in Proc. 22nd IEEE Conf. Soft. Eng. Educ. & Training, 2009, pp. 45-52.
- [35] M. Daun, A. Salmon, B. Tenbergen, Th. Weyer, and K. Pohl, "Industrial Case Studies in Graduate Requirements Engineering Courses: The Impact on Student Motivation," in Proc. IEEE Conf. Soft. Eng. Educ. & Training, 2014.
- [36] I. Richardson, Y. Delaney, "Problem Based Learning in the Software Engineering Classroom", Proc. 22nd IEEE Conf. Soft. Eng. Education & Training, 2009, pp. 174-181.
- [37] K. Garg, V. Varma, "A Study to the Effectiveness of Case Study approach in Software Engineering Education", Proc. 20th IEEE Conf. Soft. Eng. Education & Training, 2007, pp. 309-316.
- [38] B. Penzenstadler, D. Richardson, B. Karlin, A. Cook, D. Callele, and K. Wnuk, "Using Non-Profit Partners to Engage Students in RE," Proc. of the 2014 Intl. Workshop on Requirements Engineering Education & Training, 2014.
- [39] B. Brügge, and M. Gluchow, "Towards production ready software in project courses with real clients", in Proc. of the First International Workshop on Software Engineering Education Based on Real-World Experiences, 2012, pp. 5-8.
- [40] B. Al Sabbah and S. Kowalski, "A Socio-technical Framework for Threat Modeling a Software Supply Chain," IEEE Security & Privacy 13(3), 2015, pp. 30-39
- [41] M. Palacin-Silva, A. Seffah, and J. Porras, "Infusing Sustainability into Software Engineering Education: Lessons Learned from Capstone Projects. Journal of Cleaner Production 172(20), 2018, pp. 4338-4347.
- [42] J. Rodriguez, K. Greer, and B. Shipman, "Copyright and You: Copyright Instruction for College Students in the Digital Age," The Journal of Academic Librarianship 40(5), 2014, pp. 486-491.
- [43] J. C. F. de Winter, "Using the Student's t-test with extremely small sample sizes," Practical Assessment, Research & Evaluation 18(10), 2013.

## APPENDIX A: CHECKLIST ITEMS ON THE SURVEY INSTRUMENT

A ready-to-implement version of the survey instrument, the TAP template used in class, and all data in their raw format along with analyses can be found at [26].

### **L01: Provide a Justification for the Need to Acquire a Secure Product.**

1. Assess the reason why a technology is adopted.
2. Assess team members' ability to identify risks associated with adopting a technology.
3. Assess team members' ability to resolve risks associated with adopting a technology.
4. Assess the effort in person hours to adopt a technology.
5. Assign tactics, techniques, procedures, and tools related to adopting a technology to specific team members.
6. Regularly monitor problems with using the technology and report to superiors.
7. Check for robustness of the infrastructure (technical, social, or else) to handle risk associated with technology adoption.
8. Assess implementation of mechanisms to protect information contained within the product.
9. Assess copyright of adopted technology and ensure its compliance with project requirements.

### **L05: Execute a Technology Acquisition Bidding Process.**

1. Review contract documents (e.g., Technology Acquisition Plans, or documentation provided with the technology) to ensure that team members are aware of risks associated with technology adoption and training are included and are adequate.
2. Review contract documents (e.g., Technology Acquisition Plans, or documentation provided with the technology) to ensure that team members have adequate training or prior knowledge available before adopting a technology.
3. Assess team members' ability to adequately integrate the technology.
4. Draft a Technology Adoption Plan.

### **L09: Create and Execute an Effective Problem Resolution Process.**

1. Conduct a manual review and inspection, as well as acceptance testing of adopted technology.
2. Evaluate changes in adopted technology (e.g., uncovered security vulnerabilities, new versions) and manage risks associated with them
3. Monitor and audit the systems and operations to reduce the risk of unauthorized access.
4. Monitor the adoption timeline and threats to project goals; report ahead/behind/on schedule.
5. Provide feedback on vulnerabilities of the adopted technology throughout development.
6. Formulate Impact on future milestones and replan, if necessary.

# **Short Papers**



# Learning Environment Interoperability in Software Engineering Education

Dimitri Bigler

*Faculty of Computer Science*

*Kempten University of Applied Sciences*

Kempten, Germany

dimitri.bigler@hs-kempten.de

Julian Manz

*Faculty of Computer Science*

*Kempten University of Applied Sciences*

Kempten, Germany

julian.manz@hs-kempten.de

Kevin Lee

*Faculty of Computer Science*

*Kempten University of Applied Sciences*

Kempten, Germany

kevinkim.lee@hs-kempten.de

David Fischer

*Faculty of Computer Science*

*Kempten University of Applied Sciences*

Kempten, Germany

david.fischer@hs-kempten.de

Georg Hagel

*Faculty of Computer Science*

*Kempten University of Applied Sciences*

Kempten, Germany

georg.hagel@hs-kempten.de

**Abstract**—This paper explores the significance of interoperability advancements in learning management systems (LMS) and the demand for customizable, flexible learning environments. We present a system that integrates learning resources from an LMS into a highly customizable frontend, including the embedded TaskAssessment, a programming task evaluating software. By incorporating standards such as Learning Tools Interoperability (LTI) and Experience API (xAPI), seamless interoperability across various systems is ensured. The goal is to contribute to the advancement of personalized learning experiences by learning element recommendation, automatic assessment and targeted feedback, while improving system interoperability.

**Index Terms**—interoperability, LTI, LMS, LRS, xAPI, learning elements, frontend, adaptable learning environment, learning analytics, code evaluation

## I. INTRODUCTION

As part of Software Engineering Education, our system offers the possibility to perform automated short answer grading (ASAG), as well as remote code execution and correction. It is capable of generating learning analytics (LA) for later evaluation. These powerful features require a well integrated application that seamlessly accompanies the adaptive learning system (ALS).

Interoperability between systems is crucial to enable a smooth transition and consistent experience. By unifying systems on a single application, users only need to navigate and authenticate on a central platform to access a greater variety of content and services, reducing the degree of lock-in [1]. Real-world examples, such as the concept of a metaverse and the trend towards using existing accounts for authentication across different platforms, further highlight the significance of interoperability. In addition to the importance of interoperability, there is also a demand for customizable, flexible learning environments; diverse fields ranging from business [2] to nursing [3] to agriculture [4] are adopting learning management systems (LMS) to onboard and educate their users. An LMS would likely require a high degree of

flexibility and customization to meet the unique needs of each sector, as well as seamless integration with existing systems and workflows. In order to increase the flexibility and customization of such systems, other publications explore the integration of external tools using technologies such as Learning Tools Interoperability (LTI) [5]. In the field of learning technology, overcoming the challenges of integration is crucial to ensure seamless interoperability between diverse systems. The use of standards such as Experience API (xAPI) and LTI can be instrumental in this effort, offering solutions that exceed the limitations of individual LMSs. By using these standards, institutions can more easily migrate to different LMS platforms or integrate new technologies without extensive redevelopment or customization. Despite the benefits of xAPI and LTI, challenges remain in implementing and configuring these standards within an LMS environment. In [5] the complexities associated with LTI implementation are highlighted, including configuration issues and ensuring compatibility with different LMS platforms. Overcoming these challenges requires careful planning and strategic execution, often involving collaboration between developers and LMS administrators. By pursuing these objectives, this paper seeks to advance the development of adaptable, personalized learning environments. Furthermore, we describe the implementation of learning elements from the LMS into the frontend and demonstrate how these components are seamlessly integrated, using interoperable technologies such as xAPI and LTI.

## II. RELATED WORK

With widespread disruption to traditional education and training due to social distancing measures and lockdowns during the COVID-19 pandemic, institutions and organizations have turned to online learning platforms to ensure continuity in education [6], [7]. Learning platforms can integrate LMSs, which are software applications used to administer, deliver, track and manage educational content and learning

experiences. Moodle, for example, is one of the most widely used LMSs in higher education worldwide, offers great cross-platform compatibility and runs without modification on any operating system compatible with PHP [8]. Initially there have been efforts to establish a common basis for integrating external services with LMSs [9]. Among these, the LTI standard has been established and is widely supported by many of such systems. The purpose of the standard is to integrate external learning tools in a unified way, while providing certain control over those tools when needed [10]. LTI offers a set of bundled services called LTI Advantage. The name and role provisioning service supplies learning tools with information about the user's role (e.g. student, tutor, administrator) in the LMS and whether the user has previously accessed the tool. xAPI is a technical specification licensed under Apache-2.0 and standardized in [11], which provides a basic overview of version 1.0.3. Developed by the Advanced Distributed Learning (ADL) initiative, the concept emerged from the Sharable Content Object Reference Model (SCORM) and is the result of open source collaboration. The technical specification is widely used as a comprehensive guide to the rules a system must follow to utilize xAPI and a Learning Record Store (LRS). An LRS is defined in [11] as a server that accepts, persists and provides access to learning records. Retrieval and storage of xAPI statements is the main function of the learning activity repository and can therefore be used to record and access data from experiments, often in a learning context. The described technologies LMS, LTI and xAPI often occur in combination to achieve various goals. The study in [12] presents a framework for an adaptive learning system designed to introduce adaptivity into an LMS. More recent publications have used LTI in addition to xAPI to achieve a more flexible implementation in an LMS [13]. Other publications additionally use an LRS to analyze the user data and have the ability to enhance the adaptivity of the system [14], [15]. It appears that multiple technologies and systems are working together not only to collect but also to store, process and interpret the growing volumes of data. The insights from this related work lay the foundation for our approach of using LTI, xAPI and an LRS to create a flexible and customizable learning environment that extends the functionality of the LMS.

### III. METHODOLOGY

Our approach builds upon existing literature by integrating a frontend and backend component with standards such as LTI and xAPI to extend the functionality of LMSs. This model enhances the user experience (UX) and functionality within the LMS ecosystem.

#### A. Preparation

Before our system can be used, several preparatory steps must be taken. First, the LMS needs to be set up by creating courses and adding learning elements. In addition, student accounts must be created and enrolled in the necessary courses.

User accounts, course information and learning element information are then stored in our underlying backend DB, PostgreSQL.

#### B. Working with the system

Students begin their interaction with our system by navigating to the domain<sup>1</sup> and clicking the login button enabled by the LTI connection. They are then redirected to our LMS<sup>2</sup> for account authentication. Upon successful authentication, the user is redirected to our frontend. On first login, users are presented with a short form of the Index of Learning Styles questionnaire [16]. Accompanied by a brief explanation, students are encouraged to complete the questionnaire, which consists of 20 questions designed to identify their learning style. Submitting the questionnaire triggers the calculation of the learning element sequence (learning path) in the backend. For each topic in the course, different machine learning algorithms can be used to estimate learning paths. The user can then select a course, view the available topics and choose any topic to begin. Within each topic, the suggested learning path is displayed, as shown in Fig. 1. The advised direction for students to attend learning elements is from top to bottom, following the presented learning path. However, students are not constrained by the system's recommendation and can engage with topics and instructional content in any order they prefer. This behavior is analyzed with learning analytics, allowing us to evaluate the effectiveness of the chosen algorithm

<sup>1</sup><https://ke.haski.app>

<sup>2</sup><https://ke.moodle.haski.app>

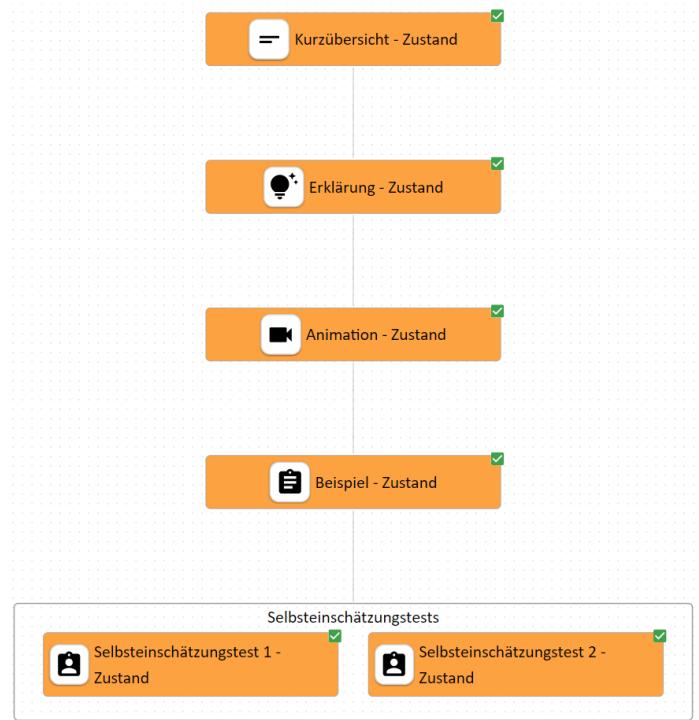


Fig. 1. Frontend showing a topic's learning path

and the user's satisfaction with it. When a student clicks on a learning element node in the frontend, a modal opens displaying an iframe containing the selected resource from the LMS. For example, Moodle offers a variety of activities such as H5P content, SCORM packages, questionnaires, videos and quizzes [17]. The user can then work on the element, completing and saving their answers. When the user meets the completion conditions for the activity, the node receives a green checkmark to indicate that the element is done and the user can proceed with their learning path. After a topic has been finished, students are encouraged to provide feedback using a Moodle questionnaire. Finally, when the learning session ends, users can log out. All information regarding their learning progress is stored in the backend DB and in the LMS. For evaluation purposes, all user activity is persisted in the LRS using the xAPI standard.

### C. Overview of xAPI

xAPI is described in [11] as a technical specification to establish communication between activities, often in an educational context and an LRS via statements. The structure of a single statement consists of several fields, such as actor, verb and object. xAPI enhances interoperability between multiple sources by providing flexibility at various stages of processing learning experience data, such as composition, collection, persistence, or evaluation.

1) *Integration of logStore Plugin:* In order to gain a more comprehensive view of student behavior within the system, xAPI statements are sent from the LMS to the LRS. The Moodle LMS used in this study does not provide this functionality natively and requires an extension through a plugin. As highlighted in [18], logStore xAPI is the only stable plugin currently available through the Moodle plugin directory that allows xAPI statements to be sent to an LRS. This extension has proven to be capable of handling large amounts of data in a reasonable amount of time [19]. However, it does not provide all the data that could be gathered through Moodle. Consequently, it has been forked and extended to incorporate all default events tracked by Moodle<sup>3</sup>. An additional private key authentication method has been added to allow the plugin to communicate with LRSs that don't support username and password authentication.

2) *Integration of custom xAPI package:* To enable interoperability between our frontend and the LRS, a way to send xAPI statements had to be established. The existing xAPI.js library [20] was extended into a separate package for seamless integration. To connect to the LRS, the overhead is managed in a setup file that requires the data store system URL, credentials and xAPI standards version. The xAPI statements are created automatically with information from the browser, such as the current URL and language settings. Additionally, the translation file, the folder structure, the current date and information from the config file like the URL to our custom verb wiki, the frontend version and the URL to our GitHub<sup>4</sup>

<sup>3</sup><https://github.com/HASKI-RAK/Moodle-xAPI-Plugin>

<sup>4</sup><https://github.com/HASKI-RAK/HASKI-Frontend>

are used to set the dynamic statement entries. Before a statement is sent, it is created by the hook's provided functionality using the component name, a verb and a function to translate page names. The function fetches the user id, uses the router location and the component id. To integrate this package into our project, the setup file must be configured correctly. Frontend elements that use the functionality import the hook with their id and a default component set. The function to send the statements can then be called within an overridden action of the component, such as "clicked", "closed" or "changed". The parameters are the appropriate verb and the current project file path. A collection of custom verbs is hosted on the wiki via a documentation generator for frontend and Moodle usage. The overridden components can be used as defaults to ensure that a statement is automatically sent to the LRS whenever a user performs an action.

3) *Integration of the LRS:* The LRS, as described in [11], offers a RESTful web services API. Supported requests include PUT, POST and GET to store single and multiple requests as well as fetch any number of statements from the LRS. We use the open source SQL LRS developed by Yet Analytics as a data store solution, which is documented in [21]. It is based on SQL and supports multiple database management systems (DBMS) such as SQLite and PostgreSQL. Our system is deployed on a server as a Docker container to persist all statements to a PostgreSQL DB. These statements can be queried directly or accessed via HTTP request. For a simpler view, all learning records are processed by the LRS itself.

### D. LMS elements in the frontend

1) *Integration and Authorization of LTI:* To further enhance compatibility with other frameworks and services, our architecture integrates the LTI standard at two occasions: Both the backend and the TaskAssessment are integrated into the LMS. The frontend concludes the process by accepting the nonce as a parameter on a predefined route. It uses this nonce to obtain a cookie from the backend which includes the JSON Web Token (JWT) as well as the expiration time. The expiration time is for UX purposes only, as the frontend can prompt the user to re-login when the cookie expires. For enhanced security, the cookie has the secure, HTTP only and same site *lax* property. As of now, the LTI standard struggles with evolving cookie policies introduced by browser standards [10]. While there is no official solution yet, we tackle this issue ourselves without interfering with the standard. Therefore, a strict constraint on the deployment of frontend, backend and LMS is that they run on the same top level domain. This allows LTI and LMS cookies to be shared. While this seems restrictive, it has multiple benefits. Learning elements from the LMS can be displayed in an embedded frame in the ALS without having to re-authenticate or make unsafe changes to the LMS configuration, allowing it to be used in an iframe.

2) *Embedding LMS learning elements in the frontend:* The learning elements are primarily delivered through the LMS, but are intended to be viewed via the frontend. Consequently,

when users interact with a node representing a specific learning element, a modal is displayed containing an iframe that links to the corresponding activity in the LMS. To maintain a seamless UX within the site, specific styling rules are applied to hide navigation elements within the iframe. These are implemented in a custom plugin<sup>5</sup> based on one of Moodle's standard themes. The user journey typically involves logging into the frontend, progressing through their personalized learning path and eventually encountering a TaskAssessment assignment. Upon opening the exercise, the user can work on the programming task, submit their code and receive individualized feedback. After completing the coding exercise, the user closes the modal and continues navigating within the frontend interface.

#### E. System design and interoperability

Our architecture, shown as a UML component diagram in Fig. 2, displays numerous components and both providing and requiring interfaces. One of these is TaskAssessment, a component consisting of a frontend and backend solution for creating programming exercises with custom text entry. Additionally, the coding exercise tool provides feedback on task completion. Two required dependencies determine the use of our TaskAssessment: A connection to an LRS via HTTP requests to send xAPI statements about user actions and an integration into the LMS as an LTI activity. The result is that students can use the programming task tool like any other LMS activity without ever leaving the platform. Another point of seamless integration is the automatic sending of xAPI statements generated by user events to an LRS. As previously stated, we use an extended version of the logStore xAPI plugin in Moodle for this purpose. The LMS provides and requires interfaces to the backend. These are used for LTI authorization, like OAuth, by sending an HTTP request with user information and receiving a token. The backend persists the collected data to a DB, in our case PostgreSQL and computes an adaptive learning path for each individual student. The learning record data can be accessed by the backend through a required interface to the LRS via xAPI REST requests. Further, a provided and a required REST interface is needed for communication with the frontend, which receives and authorizes the user data and displays the sequence of learning elements of individual users. This is implemented using the LTI specification, meaning that the TaskAssessment activities are visible within the frontend after they have been processed by the backend and the LMS. A provided interface handles user logins or triggers learning path recalculation by sending HTTP requests to the backend. The frontend itself is a flexible and highly customizable single-page application that displays all the learning activities in a learning path assigned to a specific user. This is where users complete programming tasks and all events generated by users are sent to the LRS via HTTP request with xAPI data to a provided interface. Finally, the learning activity repository persists all collected learning records in a DB, which in our case is also a PostgreSQL. This

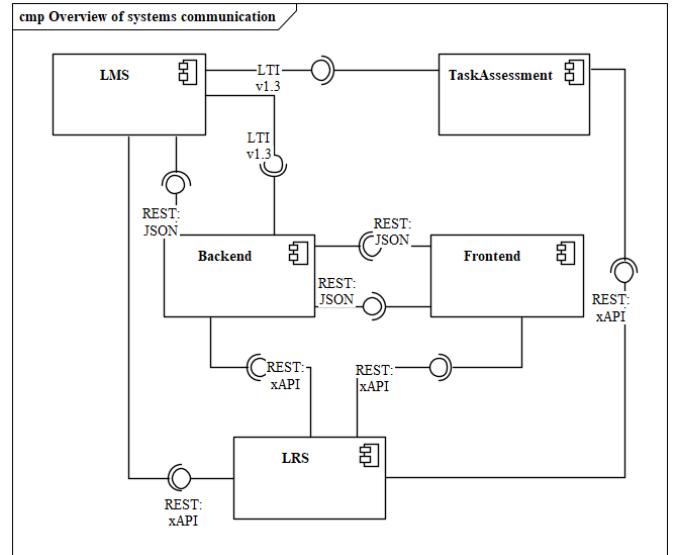


Fig. 2. Overview of our systems communication

LRS can be accessed by different systems via HTTP requests to get the data for further processing and interpretation.

#### IV. DISCUSSION

Our system has numerous advantages over traditional LMSs. Its frontend offers great flexibility, allowing developers to customize the user interface or user dashboard, implement restrictions to meet given requirements and sequence learning elements within each course. A fundamental distinction from traditional LMSs is the individualized learning path provided to each user across all courses. The reason behind tailoring courses to individual users lies in the impact that course design has on learner engagement with the material [22]. The integration of our custom systems, TaskAssessment and the frontend, allows for extensive data collection, thereby enhancing the depth and accuracy of learning analytics. Regular usage of our system takes place in a software engineering course in higher education. Unlike traditional LMS solutions, which often require frequent updates and the incorporation of scripts or plugins, our system can remain stable with minimal modifications to the LMS itself, primarily through the setup of LTI with optional data collection integration. Additionally, the autonomy of our systems allows us to choose the technologies that best suit our needs, thus overcoming the limitations of relying solely on the technology provided by the underlying LMS. However, there are a number of challenges to consider. Since only xAPI statements can be sent on behalf of a generated event, we must remain realistic and not assume knowledge about activities that do not generate events. Difficulties include the development and maintenance overhead associated with additional systems such as the frontend, backend and LRS. Despite the standardization of LTI, there are hurdles to its implementation, including compatibility issues, security considerations and standards compliance, highlighting the importance of careful integration planning [5]. Therefore,

<sup>5</sup>[https://github.com/HASKI-RAK/theme\\_haskiminimal](https://github.com/HASKI-RAK/theme_haskiminimal)

the authors of [5] suggest the application of LTI version 1.3 as a requirement. URLs must also be registered for greater security. This allows the LMS to detect any redirects within its context.

## V. CONCLUSION & OUTLOOK

Our system represents an advancement in the field of LMSs, offering seamless integration of various components to potentially enhance UX and learning outcomes. It prioritizes technology independence and enhances interoperability to ensure a unified learning environment by using industry standards for data exchange and secure communication, such as xAPI and LTI. Moving forward, one of our goals is to compare the usage of our adaptive learning system with other popular LMSs. We plan to collect and transfer a wider range of learning analytics from the TaskAssessment and the frontend to dynamically adjust the order of learning elements and make our dashboards more user-friendly. In order to gather feedback automatically, we plan to integrate feedback functionality into our frontend. Finally, we intend to evaluate our system based on the collected data to determine if it increases engagement and improves learning outcomes.

## VI. ACKNOWLEDGMENT

This work was supported by the German Federal Ministry of Education and Research (BMBF), grant number 16DHBKI036 and the Bavarian State Ministry of Science and Art. The responsibility for the content of this publication remains with the authors.

## REFERENCES

- [1] D. Carney, D. Fisher, E. J. Morris, and P. R. Place, "Some current approaches to interoperability," Carnegie Mellon University, Report, Jun. 2018.
- [2] J. K. Strakos, M. A. Douglas, B. McCormick, and M. Wright, "A learning management system-based approach to assess learning outcomes in operations management courses," *The International Journal of Management Education*, vol. 21, no. 2, 2023.
- [3] E. M. Long and Q. Bellard, "Utilizing a learning management system to provide nursing professional development," *Journal for nurses in professional development*, vol. 6, 2023.
- [4] E. Firmansyah, D. Nurjannah, S. I. Dinarti, D. Sudigyo, T. Suparyanto, and B. Pardamean, "Learning management system for oil palm smallholder-owned plantations," *AIP Conference Proceedings*, vol. 2594, no. 1, p. 130013, Apr. 2023. [Online]. Available: <https://doi.org/10.1063/5.0109194>
- [5] J. P. Leal, R. Queirós, P. Ferreira, and J. Swacha, "A roadmap to convert educational web applications into lti tools," in *Third International Computer Programming Education Conference (ICPEC 2022)*, ser. Open Access Series in Informatics (OASIcs), vol. 102. Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022, pp. 12:1–12:12.
- [6] A. Almanthari, S. Maulina, and S. Bruce, "Secondary school mathematics teachers' views on e-learning implementation barriers during the covid-19 pandemic: The case of indonesia," *Eurasia Journal of Mathematics, vol. 16*, May 2020. [Online]. Available: <https://doi.org/10.29333/ejmste/8240>
- [7] M. Kerres, "Against all odds: Education in germany coping with covid-19," *Postdigital Science and Education*, vol. 2, pp. 690–694, 2020. [Online]. Available: <https://doi.org/10.1007/s42438-020-00130-7>
- [8] J. Cabero-Almenara, M. L. Arancibia, and A. del Prete, "Technical and didactic knowledge of the moodle lms in higher education. beyond functional use," *Journal of New Approaches in Educational Research*, vol. 8, no. 1, pp. 25–33, 2019.
- [9] N. Moreno and J. R. Romero, "A mda-based framework for building interoperable e-learning platforms," in *Recent Research Developments in Learning Technologies (2005)*, 2005. [Online]. Available: <https://api.semanticscholar.org/CorpusID:55722819>
- [10] Learning tools interoperability core specification 1.3 — ims global learning consortium [www.imsglobal.org](http://www.imsglobal.org). Accessed: 2024-03-19. [Online]. Available: <https://www.imsglobal.org/spec/lti/v1p3/>
- [11] IEEE Computer Society, "9274.11-2023 - IEEE standard for learning technology-javascript object notation web service for learner experience data tracking and access," New York, USA, Oct. 2023.
- [12] A. Qazdar, C. Cherkaoui, B. Er-raha, and D. Mammass, "Aelf: Mixing adaptive learning system with learning management system," *International Journal of Computer Applications*, vol. 119, no. 15, pp. 1–8, Jun. 2015. [Online]. Available: <https://ijcaonline.org/archives/volume119/number15/21140-4171/>
- [13] I. Perez-Colado, I. S. W. B. Prasetya, B. F. Manjón, I. Martínez-Ortiz, and M. Freire, "Simplifying serious games and analytics deployment in a virtual campus using lti and xapi," in *2020 X International Conference on Virtual Campus (JICV)*, 2020, pp. 1–4.
- [14] K. P. Thai and R. Tong, "Interoperability standards for adaptive instructional systems: Vertical and horizontal integrations," in *Adaptive Instructional Systems*. Springer International Publishing, 2019, pp. 251–260. [Online]. Available: [https://doi.org/10.1007/978-3-030-22341-0\\_21](https://doi.org/10.1007/978-3-030-22341-0_21)
- [15] S. Liu, K. Moura, and H. Ogata, "The information infrastructure for analyzing and visualizing learning logs in ubiquitous learning environments," in *Distributed, Ambient and Pervasive Interactions*. Springer International Publishing, 2019, pp. 408–418. [Online]. Available: [https://doi.org/10.1007/978-3-030-21935-2\\_31](https://doi.org/10.1007/978-3-030-21935-2_31)
- [16] J. Haug, D. Fischer, and G. Hagel, "Development of a short form of the index of learning styles for the use in adaptive learning systems," in *Proceedings of the 5th European Conference on Software Engineering Education*, ser. ECSEE '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 194–198. [Online]. Available: <https://doi.org/10.1145/3593663.3593675>
- [17] Plugins. [www.moodle.org/plugins/](http://www.moodle.org/plugins/). Accessed: 2024-05-21. [Online]. Available: <https://moodle.org/plugins/>
- [18] S. Judel, E. Schnell, and U. Schroeder, "Performantes xapi logging in moodle," in *20. Fachtagung Bildungstechnologien (DELFI)*. Bonn: Gesellschaft für Informatik e.V., 2022, pp. 159–164.
- [19] D. Rotelli, Y. Noël, S. Lallé, V. Luengo, and D. Pesce, "A moodle plugin for rich xapi data logging," in *Responsive and Sustainable Educational Futures*, Viberg, Olga and Jivet, Ioana and Muñoz-Merino, Pedro J. and Perifanou, Maria and Papathoma, Tina, Eds. Springer Nature Switzerland, 2023, pp. 748–754.
- [20] C. Cook, "xapi.js." Mar. 2021, accessed: 2024-03-30. [Online]. Available: <https://www.xapijs.dev/>
- [21] Yet Analytics Inc., "Documentation index," 2024, accessed: 2024-03-30. [Online]. Available: <https://yetanalytics.github.io/lrsql/>
- [22] A. Wolff, Z. Zdrahal, A. Nikolov, and M. Pantucek, "Improving retention: predicting at-risk students by analysing clicking behaviour in a virtual learning environment," in *Proceedings of the Third International Conference on Learning Analytics and Knowledge*, ser. LAK '13. Association for Computing Machinery, 2013, p. 145–149. [Online]. Available: <https://doi.org/10.1145/2460296.2460324>



# Assessing Software Development Competences Constructively Aligned in an Open-Web Format

Axel Böttcher and Veronika Thurner

*Faculty of Computer Science and Mathematics*

*University of Applied Sciences Munich*

Munich, Germany

[axel.boettcher@hm.edu](mailto:axel.boettcher@hm.edu) [veronika.thurner@hm.edu](mailto:veronika.thurner@hm.edu)

**Abstract**—To assess in a constructively aligned way those programming competences that are relevant for the professional practice of future software developers, an assessment format would be suitable where students actively program, within the integrated development environment (IDE) that they are individually used to, and with unrestricted access for researching in the web. A format that accommodates these needs well is “Bring Your Own Device, Open Book, Open Web”, where students work on their own devices against given git repositories, with full access on knowledge bases and the internet. In this work, we share experiences and suggest well established practices for executing this exam type. As well, we discuss how the ready availability of large language models impacts this assessment type.

**Index Terms**—assessment, electronic exams, computer science education, teaching methods

## I. INTRODUCTION

In the introductory programming classes of Computer Science related study programs at our institution, students quickly learn to employ state-of-the-art integrated development environments (IDE) as they are used in professional practice. We also strive to foster in our students the ability to research and critically question relevant information on discussion boards or forums – a skill set that is highly relevant in professional programming, and thus explicitly an intended learning outcome of our introductory software development courses.

To assess these skills, a classical pen-and-paper exam is a no-go, as it would violate the principle of constructive alignment [1], which demands that learning objectives, exam objectives and learning methods should be aligned with each other. For example, in our teaching we encourage students to use the functionality of their IDE to systematically generate schematic source code snippets, such as getters and setters or method stubs. Writing these code snippets manually on paper requires a different skill set – which we do not explicitly practice with our students and which is mostly irrelevant for professional practice.

Practical exams in a predefined, restricted lab setting remedy some of these problems [2], [3]. However, they still have some unresolved issues: For one thing, students may not use the identical platform in their individually optimized customization that they employed during lab sessions on

This work was supported by the German Stiftung Innovation in der Hochschullehre (StiL), <https://stiftung-hochschullehre.de/> grant no. FRFMM-455/2022. Thank you for your support.

their own devices. For another thing, in predefined lab exam settings, access to the internet and thus research possibilities are restricted. To fulfill the principle of constructive alignment, the setting for a practical exam should offer the possibilities to access knowledge bases on the internet as well.

Exam formats that allow the usage of any material, both paper-based and from the internet, are referred to as “Open Book, Open Web” (OBOW) [4]. In combination with a “Bring your own device” (BYOD) approach, the BYOD+OBOW format offers constructively aligned assessment conditions.

## II. RELATED WORK

The question of how to address competences on *all* competence levels of Bloom’s revised taxonomy of learning objectives has been investigated in [5]. The students’ perception of the validity, accuracy and fairness of lab exams in programming education is evaluated in [2].

Depending on how open a practical exam is intended to be, teachers might have to secure the technical environment students are using during the exam in order to prevent cheating and plagiarism. Tools like the *Safe Exam Browser* (see <https://www.safeexambrowser.org>) limit the environment that can be accessed from a student’s device to Learning Management Systems. Other approaches require more or less elaborate technical set-ups as described by Kurniawan et al. [3].

With the quick rise of ChatGPT and its kin of large language models, a whole new set of challenges arises in the context of open web practical exams. [6] challenged both GPT 3.5 and GPT 4 with a set of 22 programming exercises from educational practice, analyzed the results and derived recommendations as well as “don’ts” that help to foster – and assess – the students’ authentic competency development in programming.

## III. GOALS

For the practical, digital assessment format “Bring Your Own Device, Open Book, Open Web” (BYOD+OBOW assessment) that can be executed either on campus or in a remote setting, we share experiences and discuss tips and tricks as well as typical pitfalls. Specifically, we focus on the following issues:

- What technical and organizational issues have to be handled in this assessment format?

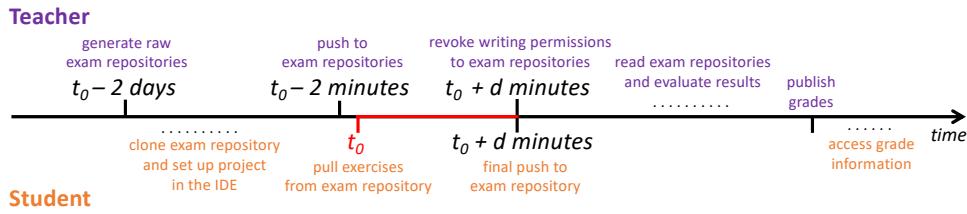


Fig. 1: Time line of exam from teacher's and students' perspective.  $t_0$  indicates the starting time of the exam,  $d$  its duration.

- Which types of exam tasks do appropriately address the different competence levels that need to be assessed in the exam?
- How are the students' deliverables evaluated, and how can this process be supported automatically?
- By what means can we create hurdles for fraud, or detect fraud if it happened?

Furthermore, if used appropriately, large language models are powerful tools that may be effectively employed in programming. How does this influence the skill set that professional software developers will need in the future, and in what way can we formally assess these skills?

#### IV. TECHNICAL AND ORGANIZATIONAL ISSUES

When setting up a BYOD+OBOW assessment, some legal, technical and organizational issues must be accommodated, such as execution mode, technical realization of student work spaces, deployment of exam assignments, as well as appropriate start end ending of exam time for each student, especially in a remote setting. To ensure that the exam will run smoothly, both technically and with respect to organizational aspects, it is advisable to take care of the technical and organizational set-up well in advance. This reduces the potential for errors in the set-up, soothes the nerves of the teachers, and also creates the time frame to allow students to familiarize themselves with the set-up prior to the actual exam situation. Fig. 1 visualizes the time line of the organizational process.

##### A. Execution Mode

BYOD+OBOW assessments can be executed either on-site or remotely. Within the lab, students can either work on their own devices (BYOD) or resort to using the lab equipment as provided by the institution. If the latter is the case, students will be provided with the standard installation that was available on the provided hardware during the lab sessions throughout the course, so that students had lots of opportunity to familiarize themselves with the environment.

If the exam is run remotely, students participate in the exam from a location of their choice, e.g., from home. In a remote exam, students usually work on their own device. However, some students cannot afford a suitable laptop or similar device. To ensure equal chances to every participant, students who do not possess an appropriate device to bring to the exam should be allowed to borrow the hardware throughout the entire semester.

While on-site exams are always conducted with supervision of students throughout the exam, remote exams can be either

supervised via video, or run in an unsupervised way. In either case, students may be legally required to initially sign a declaration that they themselves will take the exam without employing any un-permitted help, such as direct interaction with experts in the field.

##### B. Student Work Spaces

In digital lab exams, student-generated artifacts need to be safely stored in a manner that ensures that each artifact is unambiguously related to its creator, and safely available even after the end of the exam. Furthermore, visibility should be restricted so that each student can only see their own artifacts. As well, changes in artifacts after exam time has elapsed should be technically impossible. To ensure that modifications of the students' artifacts after the end of exam time would be sure to be noticed, it is advisable to attribute each stored version of the students' artifacts with a time stamp.

To achieve this, we suggest to provide each student with an individual new git repository, similar to the ones they used during the course. For our exams on Software Development with Java, we initially equip each student repository with a java project that already includes a `src`-folder, and a package that contains a predefined test class with a simple JUnit-test `assertEquals(0, ".length());` (see Fig. 2).

Once the student repositories have been created and initialized, students will receive the necessary access information for their individual repository by email a few days before the exam. This gives students the chance to set up their project environment prior to the beginning of the exam. For example, students can ensure that they have properly included the required test libraries, and that their set-up works properly. This preliminary phase helps to preclude most technical problems prior to the actual exam time.

##### C. Deployment of Exam Assignments

At the beginning of the exam time slot, both the exam assignments and any initially required source code needs to be provided by the teacher. This process is highly time critical: For reasons of fairness (and even legal requirements), it is required that either all students start working on the exam at the same point in time, or that every student is allowed exactly the same time span to work on their exam. So far, we gathered experience with two alternative approaches:

- 1) Well in advance of the exam date, the exam assignments and initial source code are pushed into the students' repository as a strongly encrypted zip archive. At the

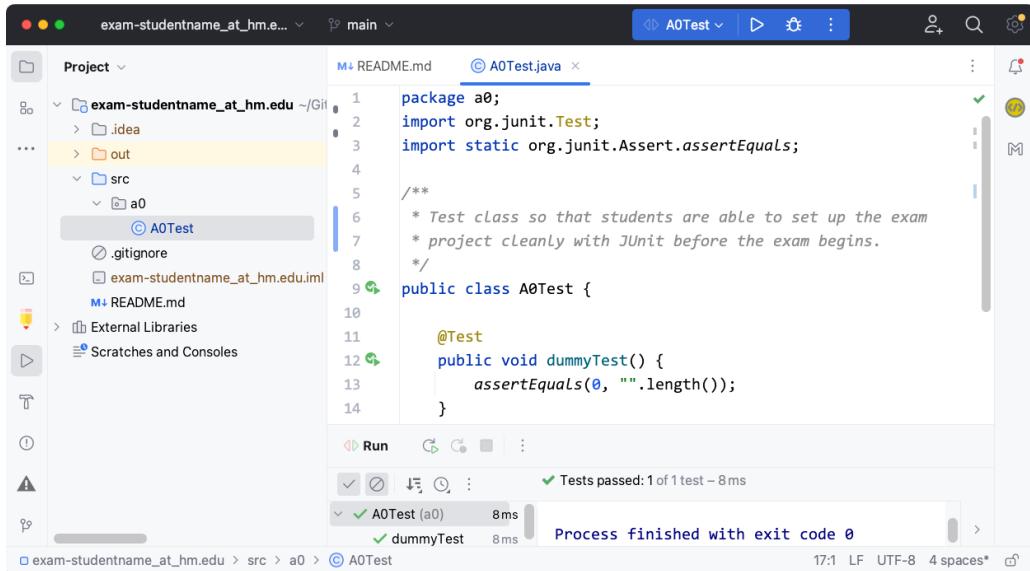


Fig. 2: Initial structure and content of the provided students' repositories.

beginning of the actual exam time, students receive the password required to decrypt and unpack the archive.

- 2) Right at the beginning of the exam time, exam assignments and initial source code are pushed (e. g. via shell script) into each of the students' repositories.

Trial runs of this process revealed that in the variant based on the zip archive, many students struggled massively with decompressing the zip archives properly.

In contrast to this, the second variant which pushes the required material via shell script into the students' repositories worked well and was a lot less error prone on the students' side. However, in the case of acute network problems, this approach is susceptible to difficulties.

#### D. Starting the Exam

To ensure equal exam conditions for every participant, we have to ensure that the time frame for processing the exam is of equal length for each student. This can be achieved by either strictly synchronizing exam time across the cohort, or by ensuring individual time frames of equal length.

In the variant where exam assignments are distributed as an encrypted zip archive, the beginning of the exam processing time corresponds to the point in time at which students receive the password that is required for unpacking and decoding the zip archive.

Note that in variant 2, where the exam assignments are pushed into each student's repository right at the beginning of the exam, the assignments will *not* be available to each student at exactly the same time. Therefore, the exam supervisors need to ensure a synchronized beginning of processing time for each student in their sub cohort.

#### E. Handling the Students' Artifacts throughout the Exam

During the exam, student generated artifacts are *not* automatically saved. Instead, students need to actively commit and

push their created artifacts into their respective repository.

#### F. Ending the Exam

Provided that the beginning of exam processing time is clearly defined and documented, it is possible to specify the exact ending of the exam processing time as well. At the end of the exam processing time, the supervisor withdraws the students' access rights for writing in their repositories via script using the git server's API.

### V. IDENTIFYING APPROPRIATE TASK TYPES FOR THIS ASSESSMENT FORMAT

Constructive alignment requires that exam tasks correspond to the learning objectives of the course. We employ Bloom's taxonomy in its revised form [7] to differentiate learning objectives on different levels of expertise. As our courses usually address learning objectives on each of these levels, our exams should cover each of these levels as well.

#### A. Lower Competence Levels

In OBOW assessments, exam tasks on levels 1 (remember) and 2 (understand) have to be designed in a way that makes it difficult for students to find the answer by a simple search of the internet. An example for an anti pattern would be to ask for the definition of a specific concept. Instead, we recommend to assess the skill of correctly using and understanding the terminology of technical concepts by incorporating the terminology into the task description.

Examples for learning objectives on these lower competence levels would be:

- *Level 1 (Remember)*: Students write down the concrete syntax of a specific programming construct, while adhering to the established syntax conventions.
- *Level 2 (Understand)*: Students explain in their own words the meaning of a specific programming construct.

An appropriate task, for which the solution cannot be simply researched in the Internet, would be “Implement a class method that ...<<description of very simple functionality>>. Adhere to syntax conventions in your implementation.” To solve this task correctly, students need to understand what a class method is, and need to know the syntax conventions.

### B. Intermediate Competence Levels

Intermediate competence levels 3 (apply) and 4 (analyse) address the students’ skills to schematically transfer what they learned into a new context. Especially on level 3 and in introductory classes, tasks tend to be rather straight forward – and thus can potentially be easily solved by ChatGPT.

Examples for learning objectives on these intermediate competence levels would be:

- *Level 3 (Apply)*: Students schematically specify basic test cases as unit tests.
- *Level 4 (Analyze)*: Students examine which normal and edge cases need to be considered within a given task, cluster these into appropriate equivalence classes and specify corresponding test cases.

A corresponding exam task provides productive source code and asks students to validate it by implementing unit tests.

### C. Higher Competence Levels

Competences on the higher skill levels 5 (evaluate) and 6 (create) require problem solving skills on a more advanced level than a mere schematic “3: apply”. However, especially in introductory classes, problem complexity needs to be sufficiently low to be solvable for novice students. Examples for learning objectives on these higher competence levels are:

- *Level 5 (Evaluate)*: Students systematically decide which control structure is best suited for implementing a specific requirement, and why.
- *Level 6 (Create)*: Based on a given requirements specification for a simple problem, students design an algorithm and implement it using appropriate control structures.

A corresponding exam task describes the functionality that students need to implement. An example would be:

“Implement in package a2 a class with a suitable class method with a parameter of type `String` and return type `String`. The method switches the initial characters of the two words passed in the parameter `String` maintaining the character’s case at the respective position. Table I provides examples that can be used for tests.”

TABLE I: Test data for the requested functionality.

method’s argument	return value
“litigating more”	“mitigating lore”
“Bad salad”	“Sad balad”
“bunny Rabbit.”	“runny Babbit.”

To ensure that solutions cannot be easily researched, it helps to formulate the task description without using obvious keywords that would be directly searchable, such as “sort” or “palindrome” (or “Spoonerize” in our example).

## VI. EVALUATING THE STUDENTS’ RESULTS

At the end of the exam, student artifacts are available electronically via git. For effectively grading the students’ solutions, we recommend to extract them from their individual repositories into a single work space project of the supervisor, and to process them there as a batch. We copy them into packages, whose identifiers reflect the students’ names.

In lab assessments where students work on git repositories, the assessment of student-generated artifacts can be at least partially automated. Tasks that request students to implement some algorithm can be validated via test cases.

In some cases we leave the freedom of naming classes and methods to our students (see Section V), so that it is not too easy to find solutions via chatGPT or internet searches. For these exercises, we always found easy ways to adapt our tests via script programming to match the identifiers that students have individually chosen in their respective source code.

On the other hand, tasks that require students to write appropriate unit tests to test a given source code example can be evaluated automatically by measuring the achieved rates of test coverage, corresponding to the levels of C0, C1 or Mutation Coverage that was demanded in the task description. If we ensure that each equivalence class of possible input values is covered by a corresponding unit test, we can directly map the number of successful and failed tests, respectively, to points that we award for the student’s solution.

Students whose repository is empty at the end of the exam will automatically fail. However, in this case we cannot detect whether such a student did not create any artifacts within their local working environment, or if they solved everything correctly but failed to push their artifacts into the repository. As students are requested to hand in their assignments via git throughout the whole semester, we assume that everyone who participated on a regular basis should be well able to correctly carry out this step during the exam as well.

## VII. AVOIDING AND DETECTING FRAUD

Generally, in our national higher education ecosystem, cheating in exams is in fact an issue, no matter which mode of execution is used. In a survey study conducted in 2021 by Janke [8] on 1608 students of different types of higher education institutions in Germany, 31.7% of the participants admitted on having tried to cheat during on-site exams, as compared to 61.4% during remote exams. Here, “cheating” comprised both the attempted or actual usage of unauthorized supplementary material and the aid of other people.

In the past, “Open Book, Open Web” assessments permitted to use any kind of written or digital resource, but explicitly excluded being aided by other persons. Thus, many classical fraud strategies from latter days, such as using cheat sheets, are simply not an issue any more in this setting.

So far, in our previous BYOD+OBOW assessments, we did not detect any plagiarism in student artifacts. To automatically check for duplicate solutions, we run some post processing using the system MOSS (“Measure of Software Similarity”, see <https://theory.stanford.edu/~aiken/moss/>).

As well, our exam supervisors did not observe any suspicious behavior, neither on-site nor in virtual exams, that indicated any improper communication attempts among the participating students. However, direct communication attempts are somewhat harder to detect, especially in the face of modern communication technology.

In the era of large language models, we need to discuss whether it is permissible to use ChatGPT and its kin. If so, we need to define task types and assessment procedures that still assess the students' authentic competences. If not, we need to establish assessment settings and formats that effectively exclude ChatGPT, or identify means to effectively detect illegal use of ChatGPT in student artifacts [6].

To make it more difficult – or at least time consuming – for students to enter entire task descriptions into search engines or front ends of Generative AI tools as prompts, we hand out the task sheet of the exam exclusively on paper. Thus, if students want to feed the task description to ChatGPT and its kin, they at least have to invest the time to manually type it in.

## VIII. META-ANALYSIS OF STUDENTS' PERFORMANCE

After the exam, we can perform some analysis on the student generated artifacts, to gain some insights at a meta level on our cohort's performance. For example, we analyzed whether students are using git as a version control tool in a professional way. To this end we plotted histograms for the number of commits performed by students, differentiating between successful and failed students as shown in Figure 3.

This evaluation does *not* reveal a clear correlation between the number of commits and the individual outcome of the student's exam result. Thus, the hypothesis that regular commits in short cycles (i. e. at least one commit per exercise – three in our case) would reflect a somewhat professional use of this tooling, cannot be upheld.

Instead, looking more closely into the commit history and underlying changes in source code of the person that failed the exam after 13 commits in Figure 3 b), revealed that this person was adding comments to the source code that indicated results of their research on the web and of using ChatGPT. A deeper dive into this code revealed that this person was clueless.

Noteworthy is the leftmost column in Fig. 3 a), indicating a student who didn't commit *anything* during the exam.

## IX. RECOMMENDATIONS AND CONCLUSIONS

Overall, BYOD+OBOW assessments are well suited to assess students' programming competences in a constructively aligned manner. In the face of evolving large language models, more effort has to be invested into creating tasks that extend the “abilities” of these systems.

In the future we will extend our tool box of task types that are suitable both for practical exercises and for automated grading (or efficient manual grading) in exams. For example, to assess students' unit testing skills, we might ask them to hand in screenshots documenting the coverage achieved.

Analyzing the evolution of solutions submitted by students to git repositories can provide valuable meta-insights into

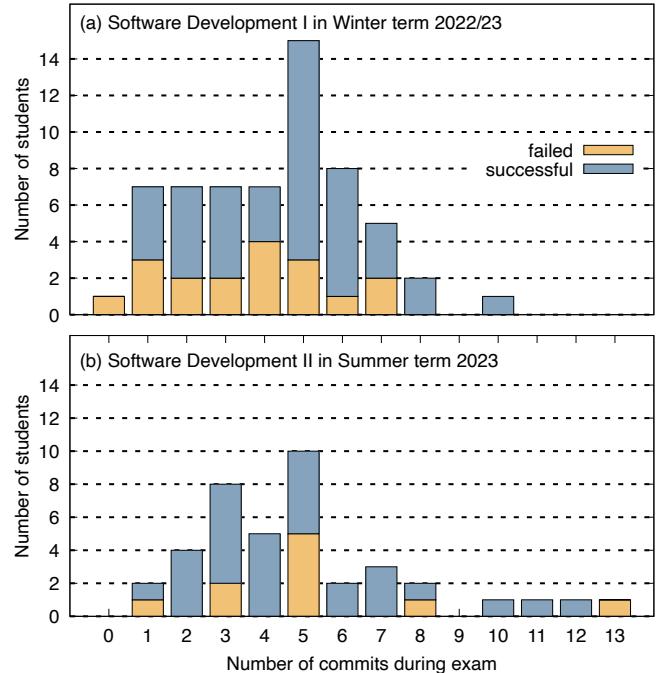


Fig. 3: Histograms of students' numbers of commits of the same cohort during exams of 1st and 2nd semester.

students' behavior and skills. For instance, we will analyze the length of time intervals between commits, as well as progress patterns between the commits of an individual student. This will help us to teach good practice patterns for the systematic use of commits – not only during an exam, but also in professional programming.

## REFERENCES

- [1] J. Biggs and C. Tang, *Teaching For Quality Learning At University*, ser. SRHE and Open University Press Imprint. McGraw-Hill Education, 2011.
- [2] S. Hammer, M. Hobelsberger, and G. Braun, “Using laboratory examination to assess computer programming competences: Questionnaire-based evaluation of the impact on students,” in *IEEE EDUCON*, 2018, p. 355–363.
- [3] O. Kurniawan, N. T. S. Lee, and C. M. Poskitt, “Securing bring-your-own-device (byod) programming exams,” in *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*, 2020, p. 880–886. [Online]. Available: <https://doi.org/10.1145/3328778.3366907>
- [4] D. Weber-Wulff, “A scaffolded, open-book, open-web exam for a first programming course in java,” in *9. Fachtagung Hochschuldidaktik Informatik (HDI 2021)*, Dortmund, 2021, pp. 175–179.
- [5] A. Böttcher, V. Thurner, and D. Zehetmeier, “Alignment of teaching and electronic exams and empirical classification of errors for an introductory programming class,” in *2020 IEEE 32nd Conference on Software Engineering Education and Training (CSEE&T)*, 2020, pp. 1–10.
- [6] J. Berrezueta-Guzman and S. Krusche, “Recommendations to create programming exercises to overcome chatgpt,” in *2023 IEEE 35th International Conference on Software Engineering Education and Training (CSEE&T)*, 2023, pp. 147–151.
- [7] L. W. Anderson, D. R. Krathwohl, P. W. Airasian, K. A. Cruikshank, R. E. Mayer, P. R. Pintrich, J. Raths, and M. C. Wittrock, *A Taxonomy for Learning, Teaching, and Assessing. A Revision of Bloom's Taxonomy of Educational Objectives*, 1st ed. New York: Longman, 2001.
- [8] S. Janke, S. Rudert, A. L. Petersen, T. Fritz, and M. Daumiller, “Cheating in the wake of covid-19: How dangerous is ad-hoc online testing for academic integrity?” *Computers and Education Open*, vol. 2, p. 100055, 10 2021. [Online]. Available: <https://doi.org/10.1016/j.caeo.2021.100055>



# Using Generative AI to Create User Stories in the Software Engineering Classroom

Allan Brockenbrough  
Computer Science Department  
Salem State University  
Salem, US  
abrockenbrough@salemstate.edu

Dominic Salinas  
Computer Science Department  
Salem State University  
Salem, US  
d\_salinas1@salemstate.edu

**Abstract**—A user story is used in agile methodology to describe functionality that is valuable to the user and may include criteria to determine if the developer has completed the story. This study investigates undergraduate computer science students using ChatGPT to create user stories from user feedback. The study compares aspects of the user stories created by students using ChatGPT with those not using ChatGPT. Are user stories written by students with AI assistance of higher or lower quality? How does the time spent writing the user story change with the use of ChatGPT? We evaluate student user stories using a modified INVEST story rating system. Evaluated user story properties include structure, independence, value, testability, and grammar. The results show that ChatGPT-assisted students produce higher-quality user stories than unassisted students. However, using ChatGPT to write user stories does not guarantee high quality. ChatGPT can fail to recognize dependencies between user feedback and create structurally incorrect user stories. We see a need for students to be trained in effectively using this tool by carefully examining AI-assisted output and making revisions.

**Keywords**—user stories, ChatGPT, GPT-Generative AI, INVEST, software engineering education, LLM

## I. OVERVIEW

ChatGPT, a generative AI and large language model (LLM) chatbot introduced in November 2022, has been changing the landscape of the software engineering classroom. Students can use generative AI to solve undergraduate programming problems [1], changing the traditional grading of class assignments. Time spent coding may decrease as AI chatbots provide potential draft solutions to coding tasks. The focus of the education of software engineers may shift toward non-coding software engineering skills such as requirement engineering, design, testing, and integration [2]. Although much of the focus of ChatGPT in the software engineering classroom has been on coding tasks, this study investigates using ChatGPT for other aspects of software engineering, namely, creating user stories.

A user story is used in agile methodology to describe functionality that is valuable to the user [3]. Traditionally, this story describes the functionality and may include a way of determining if the story is complete. The user story may be created from user feedback for a new application, feature, or changes to existing functionality. Requirements are a fundamental component in software engineering introduced in undergraduate software engineering classes [4]. Instructors

teach students to create appropriately structured user stories based on analysis of user interviews or feedback.

User stories begin with an informal description of a feature from the user's viewpoint. The goal of the user story is to describe a software feature that is valuable to the customer written in non-technical language. The developers use these stories to guide what they are building by understanding the value the story provides. The user story is typically written in the first person following a template, such as “As a <user role>, I want to <perform a task>, so I can <achieve a goal>.”

This study quantitatively analyzes user stories written by software engineering students in the classroom using ChatGPT to author the stories. This analysis addresses the research question (RQ1): “*Do students using ChatGPT 3.5 to create user stories based on user feedback improve the quality of the user stories over stories written without ChatGPT 3.5 assistance?*”

To answer this question, we examine the quality of user stories created with and without AI assistance. To determine the quality and areas of weakness, we use the INVEST methodology to score the stories [5]. By investigating ChatGPT’s strengths and weaknesses in assisting students in creating user stories from feedback, we aim to guide educators in using this tool in the software engineering classroom.

## II. RESEARCH METHODOLOGY

### A. Research Setup

The students in this study were undergraduates majoring in computer science. All students in this study had completed prior courses in the principles of coding, design, data structures, and algorithm analysis. The researcher presented a lecture on creating new user stories by examining user feedback. The lecture presented the user format to use in the construction of a story: “As a <user role>, I want to <perform a task>, so I can <achieve a goal>.”

We also reviewed two additional elements that provide valuable user story information with the students: the Definition of Done (DoD) and traceability of user story origin. The Definition of Done is a set of criteria used in the Scrum process to verify that a user story has been completed [6]. Traceability identifies the specific user feedback statements that are the new story’s origin. We discussed the INVEST framework for evaluating user stories [5] with students to gain an understanding of the properties of a high-quality user story.

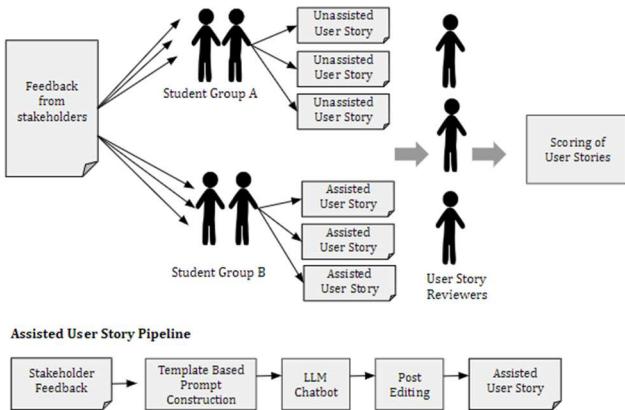


Fig. 1. Overview of methodology.

After presenting this lecture, we gave the students a task: take five fictional brief user feedback statements on change requests to an existing social media application and create the appropriate user stories to represent the feedback. We instructed the students to include a Definition of Done and identify the feedback statement(s) that originated the user story. Several of the user feedback statements provided overlapping comments. To correctly generate user stories, the students must identify common threads in the feedback and combine the appropriate feedback.

Our study also considered the impact ChatGPT had on the time students spent creating user stories. To investigate this, we gathered information on the time students spent writing user stories, both AI-assisted and unassisted. We timed the students from the moment they received the user story task until they submitted the stories to the learning management system. Contrasting the time taken by the two groups of students gives insight into the influence of ChatGPT on user story creation time.

We randomly divided the students into two groups (see Figure 1) and instructed both groups to complete the task independently without collaboration. Group A was informed to work without ChatGPT, while Group B was required to use ChatGPT.

We gave Group B the prompt for the ChatGPT request (Figure 2) to create the stories. We asked Group B students to take the response and potentially make revisions to improve the quality of the user stories.

Can you write user stories that address user feedback? You will have to analyze the feedback and create appropriate user stories. Here is a description of the application followed by the feedback. The sentences in the feedback are numbered. You can reference the sentence numbers in the user stories.  
The Posince social media application lets the user create posts and explore other people's posts. The user can make comments on a post, edit their own posts, or follow another user. The user can view a feed of posts the system presents based on an algorithm.  
The feedback is  
I do not like reading posts with offensive language.  
I would like to be able to reply to posts.  
I like making posts, but it would be great to add a music audio track to the post.  
I would like to have conversations with others in the replies to posts.  
I would like posts with offensive language to be removed or at least have a warning label.  
Write a user story for the above stakeholder feedback.  
The user story will be judged on completeness, correctness, and verifiability.  
The user story should include a definition of done.  
The user story should not include implementation details.  
The user story should not include acceptance criteria  
The definition of done should be verifiable, should not mention testing, and should be no longer than 5 items.  
The user story should refer to any sentences used to generate the user story by number.  
Spread the definition of done across the user stories appropriately.  
Structure  
As a <type of user>  
I want to <what the user wants to do>  
So that <achieved value>  
Origin <feedback sentence numbers>

Fig. 2. ChatGPT 3.5 prompt for generating user stories.

**User Story 1: Moderating Posts for Offensiveness**  
As a user, I want to have the option to report posts with offensive language, So that I can contribute to maintaining a positive and respectful environment within the application.  
**Origin:** Feedback Sentences 1 and 5  
**Definition of Done:**  
1. Users can report a post for offensive language.  
2. A mechanism is in place to review reported posts.  
3. Reported posts are either removed or appropriately labeled as offensive.  
4. The process of reporting and handling offensive content is streamlined and user-friendly.

Fig. 3. User story produced using ChatGPT with the correct structure.

As a user of the Posince social media application, I want to filter out posts with offensive language,  
So that I can have a more positive and enjoyable experience while using the application.  
Origin: 1

Fig. 4. User story produced using ChatGPT missing feedback dependencies and Definition of Done.

### B. User Story Scoring

The stories were evaluated using a revised INVEST framework. The INVEST method proposed by Bill Wake [5] considers six properties of user stories. We used the original six properties and added three more: Structured, Traceable, and Grammar. These criteria are described below.

**Independent:** The stories should not overlap in content.

**Negotiable:** The story should not be overly detailed and negotiable with the user.

**Valuable:** The story should be valuable to the user. The story should be coherently written so the value is understandable. An overly vague story or a story missing DoD is not valuable.

**Estimable:** The story should have sufficient clarity and not be overly large so that a developer can estimate the size of the work.

**Small:** The user story should be granular enough to be implemented in several weeks.

**Testable:** the user story should be testable. The Definition of Done provided as part of the user story should be specific enough that it would be possible to write a test case.

**Structured:** the user story should follow the structure of the user role, goal, and value. The story should be written in the first person. The story must include the origin of the user story by identifying the user feedback and providing a Definition of Done.

**Traceable:** the user story should reference all user feedback used to create the story.

**Grammatical:** the user story should be professionally written and free of grammatical and spelling errors.

These characteristics are scored using a binary system, with a score of one indicating that the category has mostly met or a score of zero for an unfulfilled category.

## III. RESULTS AND DISCUSSION

The students created 75 user stories, with 38 written with the aid of AI and 37 written with no assistance. A researcher not involved with scoring the stories removed any references to ChatGPT and student identity information. Using the expanded INVEST, two computer science professors independently calculated the score for each user story property, discussed the results, and arrived at a consensus score. We summed the property scores to calculate the total score for the story.

### A. Overall Performance

The collective average score for students writing user stories assisted by ChatGPT was approximately 17% higher than stories than those written without. As shown in Figure 6, the number of higher-quality user stories, those with a score of 80% or above, was substantially higher when AI was used. When AI was used, over 58% were high-quality. The

percentage of high-quality stories fell to just 30% for unassisted students.

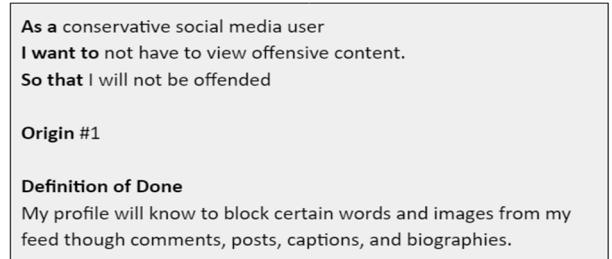


Fig. 5. Example of a user story written by a student without AI assistance.

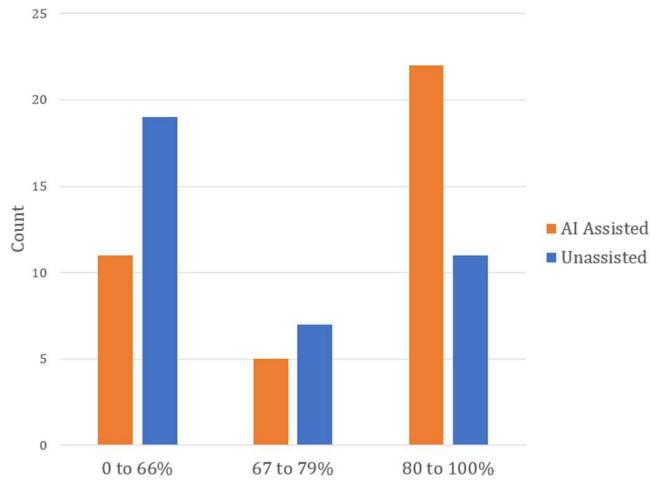


Fig. 6. Distribution of scores by students using ChatGPT or not.

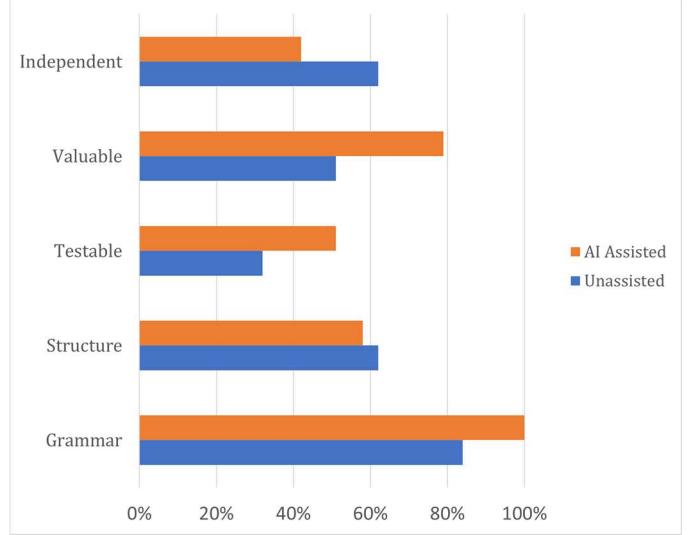


Fig. 7. Comparison of the average score for user story by characteristic.

ChatGPT had difficulty in two specific areas: (1) identifying and grouping related user feedback to create a single user story and (2) writing a Definition of Done for each story. Although an identical prompt was used, ChatGPT's answer varied, often stumbling over these two areas. Figure 4 shows an example of these failures. The user story did not use related feedback #5 and did not include a Definition of Done. This error caused lower scores in Independent, Testable, and Structure properties.

ChatGPT scored highly in all other properties, with perfect scores in several areas. The following sections discuss characteristics with lower scores in more detail.

#### B. Independent

In preparing the experiment, we carefully constructed the ChatGPT prompt (Figure 2). The prompt asks ChatGPT to take a collection of user stories, some of which are overlapping, and create a set of user stories. The created user stories must be independent, requiring ChatGPT to determine when feedback should be grouped.

ChatGPT had difficulty consistently identifying this grouping correctly. Table 3 shows that students who did *not* use ChatGPT outperformed those who did (characteristic “Independence”). This is also reflected in Figure 7, showing AI scoring lower in Independence.

If a student examines the AI output, it is easy to detect that ChatGPT did not group the customer feedback. This may indicate that students using AI assistance submit the response with little analysis or revision.

#### C. Structure, Definition of Done, Testable, and Valuable

The prompt requested that a Definition of Done be written for each user story. ChatGPT had difficulty with this request and occasionally created a single Definition of Done for all the stories with limited details. This is reflected by the poor performance of 58% of the characteristic “Structure” (Figure 7, Table 1). The single Definition of Done for all stories also caused Testable issues due to the lack of sufficient detail to create test cases.

Students also struggled with writing the Definition of Done by failing to provide enough detail or occasionally writing one Definition of Done for all stories.

#### D. Traceability

The ChatGPT consistently successfully reported feedback items used in forming the user story. Although ChatGPT may fail to group related feedback items, the item origin was always reported (examples in Figures 2 and 3). Both the AI-generated and unassisted user stories reported traceability with a high degree of accuracy of 100%.

#### E. Grammar

ChatGPT excelled in creating grammatically correct user stories with a perfect score in this category. The performance of unassisted students was 11% lower in this category.

#### F. Negotiable, Estimable, and Small

For the remaining characteristics of “Negotiable,” “Estimable,” and “Small,” both groups of students scored

highly with little differences in scores. Table I summarizes the scores for all user story evaluation categories.

#### G. User Story Task Completion Time

Students using ChatGPT showed a notable time advantage in creating user stories from feedback compared to their unassisted counterparts. On average, students who used AI assistance were 38.1% faster than those who did not. Figures 8 and 9 depict the time taken by students to write all user stories.

TABLE I. SUMMARY OF AVERAGE SCORES BY CHARACTERISTICS

User Story Characteristic	Unassisted	AI-Assisted
Testable	32%	58%
Structured	62%	58%
Independent	62%	42%
Valuable	51%	79%
Estimable	43%	71%
Grammatical	84%	100%
Small	43%	68%
Traceable	100%	100%
Negotiable	100%	100%

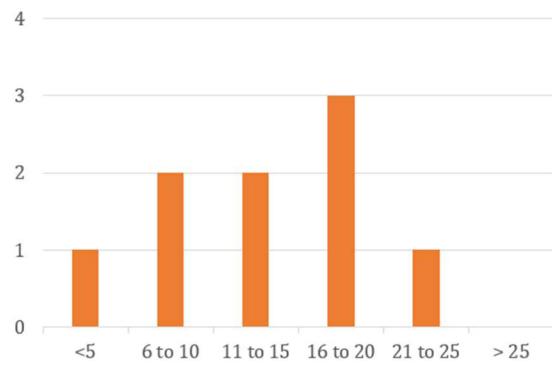


Fig. 8. Number of stories completed in minute intervals with AI assistance.

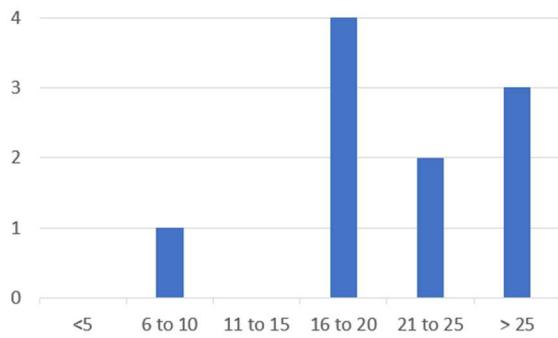


Fig. 9. Number of stories completed in minute intervals without AI assistance.

#### IV. CONCLUSION AND FUTURE WORK

This study found that students who were assisted by ChatGPT 3.5 produced higher-quality user stories than unassisted students. However, ChatGPT did not consistently produce high-quality user stories. Giving repeated identical user story prompt requests to ChatGPT containing user feedback and asking for appropriate user stories including Definition of Done, gives differing results. The Definition of Done was sometimes not separated by the user story and failed to contain proper details. ChatGPT occasionally failed to group user feedback and instead created overlapping stories. More often, however, the AI-assisted response was of high quality. This points to the importance of educating students on how to evaluate user stories and applying this analysis if AI is used to create an initial draft. Students should use AI carefully, analyzing AI-assisted output and making thoughtful revisions.

The focus of our study was to understand the usefulness of AI in helping students transform a collection of user feedback statements into multiple covering user stories. The inconsistency of ChatGPT in properly structuring a collection

of user stories suggests that future studies might consider a more straightforward user story task. ChatGPT could be given a set of related user feedback statements and asked to create a single user story with the Definition of Done. This may be a more appropriate use of AI for user story creation and provide more insight into the usefulness of this technology for students.

#### REFERENCES

- [1] E. L. Ouh, B. K. S. Gan, K. Jin Shim, and S. Włodkowski, "ChatGPT, Can You Generate Solutions for My Coding Exercises? An Evaluation on Its Effectiveness in an Undergraduate Java Programming Course," *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1*, Turku, Finland, 2023, pp. 54–60.
- [2] M. Daun and J. Brings, "How ChatGPT Will Change Software Engineering Education," *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1*, Turku, Finland, 2023, pp. 110–116.
- [3] M. Cohn and K. Beck, *User stories Applied: For Agile Software Development*, Addison-Wesley, 2011.
- [4] I. Sommerville, *Software Engineering*, Pearson, 2015.
- [5] B. Wake, "INVEST in Good Stories, and SMART Tasks," XP123, <https://xp123.com/articles/invest-in-good-stories-and-smart-tasks/> (accessed Jan. 3, 2024).
- [6] K. Schwaber and J. Sutherland, "The Scrum Guide the Definitive Guide to Scrum: the Rules of the Game," 2013. Available: <https://scrumguides.org/docs/scrumguide/v1/Scrum-Guide-US.pdf>.



# Adapting Computational Skills for AI Integration

Hanya Elhashemy

*Technical University Of Munich  
Massachusetts Institute of Technology  
Munich, Germany  
hanya.elhashemy@tum.de*

Harold Abelson

*Massachusetts Institute of Technology  
Massachusetts, USA  
hal@mit.edu*

Tilman Michaeli

*Technical University Of Munich  
Munich, Germany  
tilman.michaeli@tum.de*

**Abstract**—In today’s data-driven world, the importance of data literacy is paramount. However, software engineering education has not adequately addressed integrating comprehensive data science curricula, leaving students ill-equipped for the future of artificial intelligence (AI), which is built on the foundations of data science. This gap is exacerbated by the lack of tailored courses and the intimidating nature of existing tools for beginners. Consequently, students often miss out on essential skills like data cleanup, real-world application of machine learning (ML) algorithms, and the integration of big data in software products.

This paper addresses these challenges by proposing a novel approach to applied data science for software engineering students. We argue for a shift from traditional algorithm-focused teaching to a curriculum emphasizing real-world problem-solving, leveraging data science techniques. By empowering students to define and tackle their own data-driven projects, we aim to increase motivation, enhance data literacy, and instill a data-thinking mindset in future software engineers to prepare them for the AI world.

Overall, this paper contributes to the advancement of software engineering education for young learners by offering a comprehensive framework, the data action educational framework (DAEF), and a data science toolkit that enables DAEF by empowering learners to create original data-driven mobile apps.

**Index Terms**—data thinking, data science, software engineering education, computational action

## I. INTRODUCTION

Observing the current trends in software engineering, we noticed many software applications integrating ML, AI and generative AI (Gen AI) in their core business logic. Many people experience AI and interact with it without knowing how it functions [9]. All this increases the demand for software engineers who grasp the concepts of ML and AI to be able to efficiently modify the design of their software applications and accommodate the required infrastructure of ML models [1].

We argue that software engineers of the 21st century must understand AI’s limitations and potential to be able to leverage its power responsibly in their software applications. In the industry, this need created a new role besides software engineers and data scientists, referred to as ML engineers. These engineers often collaborate with data scientists, software engineers, and domain experts to understand business requirements and develop ML solutions that address specific needs.

In University, when raising the new generation of software engineers in the dawn of AI we need to ensure that the students

are equipped with data thinking [2]. Data thinking integrates computational thinking, statistical thinking, and domain thinking [2]. It provides software engineers with the cognitive process or a mindset to approach problems with a data-centric perspective [3], [8]. While a traditional software engineer sees “name”, “birthdate”, and “address” as user profile data stored in a database and used for login, a data-thinking software engineer sees beyond and understands the potential of this data in generating business value (e.g., customer segmentation, predictive modeling, geospatial analysis, personalization, customer insights, etc.) and would design their system differently to cater for these features.

In traditional software engineering curricula, students usually have one or two courses teaching them about working with data: (1) Algorithms and Data Structures Course and (2) Database Theory Course. In today’s data-driven world, these courses no longer provide enough skills for AI integration. They mainly provide knowledge on collecting, storing, accessing, or organizing data but not knowledge on extracting insights from gathered data and identifying patterns [4]. To leverage the power of AI, software engineers need to have data-thinking skills to be able to identify innovative use cases for the application and integration of AI in software products.

## II. RELATED WORK

We found several “Intro to Data Science” courses (often targeting graduate students); however, their educational approaches prioritize content over learner-centered design, focusing on what students should learn rather than what they want to learn. They are often limited to theoretical content and algorithmic understanding, not enabling students to engage in hands-on, data-driven projects, and missing the opportunity for students to acquire terminology and skills naturally through practical experience. The closest to our work is Acuña’s project-based teaching approach for a data science course to support software engineering students [5]. The main deliverable of his course’s project is a data analysis workflow. Adding to his approach, we empower students to build tangible software products to solve a personally relevant real-world problem of their choice while experiencing activities across the entire data lifecycle [6]. Notably, existing curricula lack sufficient coverage of data cleanup, a fundamental aspect of data literacy crucial for ensuring the quality and accuracy of data-driven insights [7]. Educators often provide students with

the “perfect” (most fitting) dataset example to focus on algorithmic understanding. Through our approach, which enables students to work on real-world problems, they experience what real-world data looks like and face the challenge of cleaning it.

### III. BACKGROUND & CONTEXT

Computational action [11] is a framework empowering learners to apply computational skills to create meaningful and impactful solutions to real-world problems. It involves not just understanding and applying algorithms, but also leveraging these skills to design, build, and deploy applications that address real-world problems. Computational action emphasizes the practical and transformative potential of computing, encouraging individuals to move beyond theoretical knowledge and actively engage in creating technology that can drive social change.

To prepare students for AI integration, the computational action framework should extend to enable students to realize the impact of data and support students in transforming their current computational skills to be data thinkers. We focus on five data literacy skills needed to transform students’ computational skills for AI integration based on the data literacy competency model [10].

Firstly, individuals must master data collection methodologies, encompassing qualitative and quantitative data gathering while scrutinizing sources for bias. Secondly, proficiency in data analysis entails preprocessing — data cleanup, hypothesis development, critique, and choosing appropriate tools and algorithms for different types of datasets. Thirdly, evaluating data by assessing its quality, and formulating new research questions based on analysis insights. Fourthly, effective data visualization is crucial for communicating insights to stakeholders. Lastly, ethical considerations such as privacy, data accuracy, and model misuse awareness are paramount, emphasizing the societal impact and responsible use of data-driven projects.

### IV. THE DATA ACTION EDUCATIONAL FRAMEWORK (DAEF)

We introduce DAEF, which expands upon the Computational Action Framework [11] with a focus on data-related computations, providing a sustainable data science curriculum, and empowering students to engage in meaningful data-driven projects relevant to their lives and communities. We designed DAEF for K-12 and target (but are not limited to) high school and freshmen students aged 14 to 18 who are familiar with (block-based) programming but have no data science background; this also applies to freshmen software engineering students. It enables them to learn data science concepts while building mobile apps to address real-world problems. Through iterative project-based learning, students experience an entire data lifecycle and acquire skills in problem identification, domain research, real-world data analysis, data visualization, and data-driven app development.

#### A. Concepts

The framework imparts four foundational concepts to the learner.

- 1) Defining a real-world problem: Students acquire the ability to identify significant real-world problems that hold personal relevance to them. They further develop skills to define and abstract these problems, enabling them to address and solve them using data science methodologies.
- 2) Self-acquiring knowledge of the problem’s domain: Unlike traditional teaching techniques where educators dictate the knowledge to be transferred to students, here the students themselves define what knowledge they require to solve the problem they identified.
- 3) Working with real-world data critically: Students engage with authentic, real-world data, often characterized by its “messiness” and potential errors. They learn to apply various data science algorithms to process this data and critique it.
- 4) Understanding the impact of data: Through their data-driven solutions, students observe the significant impact of data and the meaningful insights it can yield.

Students practice these concepts by developing mobile apps to solve real-world challenges, collaborating with peers, receiving feedback, and refining their apps iteratively. They gain insights into the challenges of working with real-world data, including data anomalies, errors, and ethical considerations. Students develop a deeper understanding of data science concepts through hands-on experience developing a data-driven mobile app in a software engineering setting and engaging in all phases of a data science lifecycle, including data collection, cleanup, visualization, and prediction [13].

### V. APPLYING DATA ACTION

Recognizing the borders between a software engineering and a data science degree, we do not expect software engineers to be experts in creating ML models but to know when and how they can be integrated into software projects. In the following, we describe an example of employing the Data Action educational framework.

#### A. Tools

We use App Inventor [18], a visual programming environment that enables learners to create fully functional mobile apps using a drag-and-drop interface. We extend App Inventor with a data science toolkit [14] to abstract the implementation details of ML models and allow software engineering students to focus on problem-solving as data-thinking software engineers.

The toolkit refers to a set of data science features utilizing block-based programming, accessible to students for addressing real-world problems. The data science code blocks snap together like Lego blocks, integrating different data science features in a mobile app.

Solving real-world problems requires working with real-world data, which often exhibits inconsistencies, incompleteness, errors, and varying formats, posing challenges for analysis and interpretation. To interact efficiently with ML models, software engineers must understand such challenges and learn how to prepare and clean data to get the best results from used ML models [12]. For this reason, we introduce the anomaly detection code block (see figure 1). It automatically flags potential anomalies in a given dataset and allows students to use their domain knowledge to decide whether to remove or retain them, while also providing a preliminary visualization feature for initial visual analysis and identification of out-of-context anomalies on mobile devices.

In addition, we introduce regression code blocks so that students can interact with supervised ML algorithms on a high level, produce predictions, and see the direct effect of data cleanup on their prediction results.

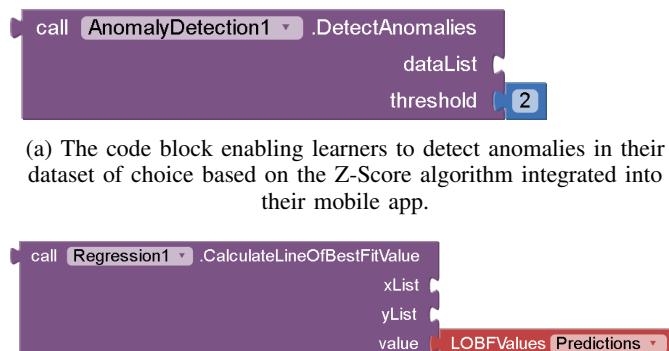


Fig. 1. Examples of code blocks from the data science toolkit integrated in App Inventor

The various data science code blocks expect as input the different columns of a dataset and as output return the prediction results in a list. App Inventor provides multiple components to store data: CloudDB, DataFile, File, Spreadsheet, and TinyDB. The dataset can also be generated through the app itself, by collecting user input or input from supported mobile sensors (e.g., light sensor, location sensor, accelerometer sensor, etc.).

### B. Milestones

While the educational framework gives students the full freedom to choose their topic of interest, we set a list of milestones the students have to reach, to ensure they experience the complete data science lifecycle. Based on the data science lifecycle, we defined the following milestones:

**MS1 Project Proposal:** We ask the students to identify real-world challenges that hold personal significance to them and that they wish to solve. As an artifact, they formulate a visionary scenario of their identified project idea and present it to their peers.

**MS2 Domain Research & Data Collection:** We ask the students to research their chosen domain and gather data supporting their project and solution idea. As an artifact, they create a dataset in .csv format. The students can either self-generate their dataset using their own collected data or they can use a publicly available dataset.

**MS3 Data Storage and Visualization:** The students explore App Inventor's data science blocks and start drafting a data and software architecture for their mobile app. They make design decisions on how to store their data and choose the most fitting data features and graph types to visualize their collected data. As an artifact, we expect them to use App Inventor to create a simple one-screen mobile app displaying their data.

**MS4 Data Cleanup:** We ask students to visually explore their data in App Inventor to identify initial irregular phenomena and spikes. The students examine these irregularities more deeply and integrate anomaly detection algorithms to identify more outliers and remove erroneous data points based on their domain knowledge. The artifacts of this milestone are a revised data set and additional anomaly detection features integrated into their mobile apps.

**MS5 Predictions:** Students integrate additional ML algorithms in their mobile apps, through which they can offer the app users insightful predictions. Students re-clean their data when necessary to optimize their prediction results.

**MS6 Additional Features:** Before the final submission of their mobile apps, we expect the students to add more features to increase the usability of their apps. These additional features do not have to be data-driven.

The students apply their projects in an agile, scrum-based setting [15], with weekly meetings to peer-review each other's project status after each milestone.

### C. Examples

We applied this approach with freshmen students. The students identified multiple real-world problems they tried to solve as part of the project proposal milestone. The problems include student debt, lack of motivation for sports, climate change, diabetes complications prevention, and more. In this section, as an example, we will present a fully working data-driven mobile app developed by a freshman student using the App Inventor data science toolkit in the context of applying DAEF.

The Diabetes Logbook App is designed to help diabetes patients manage their glucose levels. Through the app, users can log their blood glucose levels after meals, visualize their past data on a graph, and compare their levels to averages of patients with similar demographics (see figure 2).

The app automatically detects anomalies, such as mistyped data points, and provides options to remove them. Additionally, it offers personalized warnings and suggestions for self-regulating glucose levels based on real-time data inputs and an analysis of the patient's history. Users can also communicate with their doctors directly through the app in critical situations,

empowering them to take control of their health easily and conveniently.

In summary, experiencing the framework changes students' perspectives on the world and themselves, as they connect their learning to real-world issues, acquire data science skills, and develop solutions to challenges significant to them and their community. They gain awareness of data science's influence on their lives, understand the importance of data integrity, and feel empowered to create data-driven products. The framework enhances students' critical thinking, problem-solving, and data literacy skills, preparing them to navigate and contribute to an increasingly data-driven society.

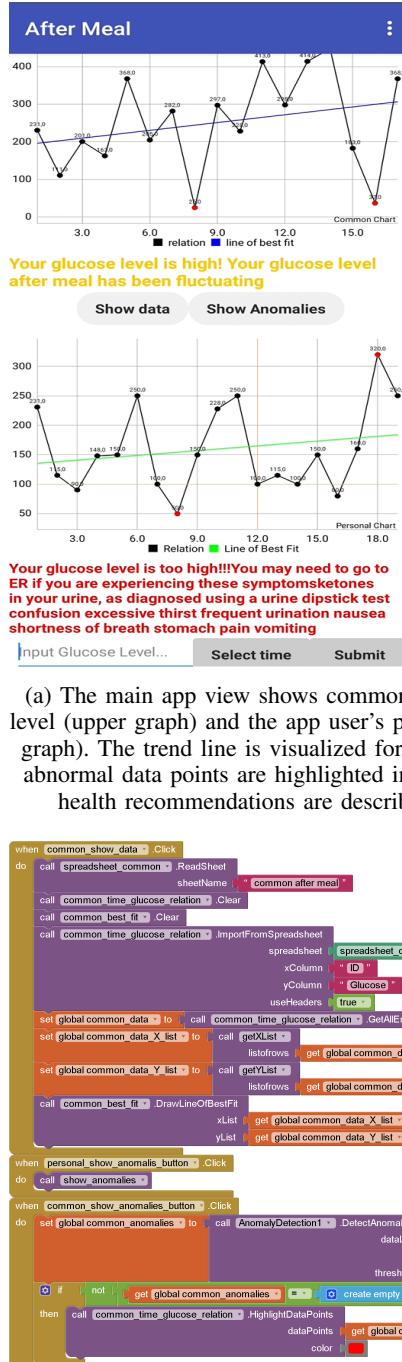
## VI. CASE STUDY

Additionally, we investigated the effectiveness of Data Action in a case study. The research questions we investigated were: (RQ1) Is DAEF effective in enabling students to recognize the impact of data science and utilize it to build impactful solutions for real-world problems? (digital empowerment) and (RQ2) Is DAEF effectively empowering students to be data thinkers (self-efficacy)?

Besides applying DAEF with freshmen students for an entire semester, we conducted a three-hour workshop with 13 students who have no prior data science experience as a research study examining the evolution of students' views before and after educational activities of DAEF. The student's views were collected through pre- and post-surveys (see 3 for exemplary items) to evaluate the effectiveness of our educational framework and App Inventor's data science blocks. During the workshop, we focused on two learning objectives of DAEF: (1) Working with real-world data critically and (2) Understanding the impact of data.

For RQ1, our findings [17] include that DAEF changed students' views on the importance of data cleanup and the effect of anomalies on their regression results. More than 45% agree that they would include data science techniques in the tech projects they create in the future. The majority believe that including data science would enhance their mobile apps (see figure 3 b).

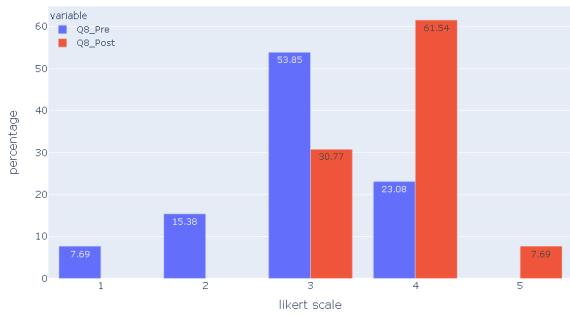
In addition, for RQ2, students reported increased confidence in their data literacy skills. We asked the participants to answer four questions before and after the workshop that assessed the students' sense of confidence in (1) data science terminology, (2) working with real-life data, (3) data visualization, and (4) data analysis. For all four categories, we found an apparent increase in participants' confidence in their data literacy skills compared to before the workshop. The majority agree or strongly agree about their data literacy abilities, with 84.62% for confidence in data science terminology, 46.15% for confidence in working with real-life data, 61.53% for data visualization, and 69.23% for data analysis (see figure 3 a for the later). DAEF empowered students to see themselves as individuals who could translate their acquired data literacy skills to solve impactful real-world problems using real-world data. Before the workshop, only around 38% of the participants



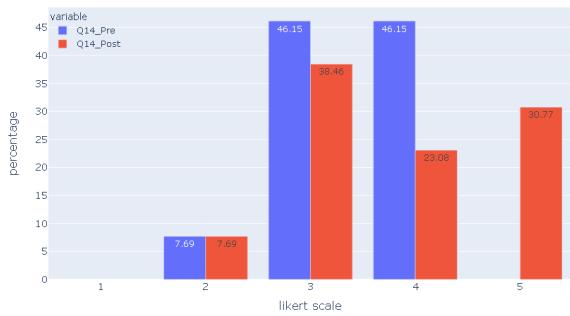
(b) Some of the app's features corresponding code blocks. The upper blocks ("common\_show\_data") are responsible for reading data from a spreadsheet, visualizing it on the graph, and drawing its corresponding line of best fit. The lower code blocks ("common\_show\_anomalies\_button") are responsible for detecting anomalies on button click, and coloring them in a distinct color (red, in this example).

Fig. 2. The main view of the production-ready Diabetes Logbook app, showcasing some of the app's features

were confident in their ability to solve real-world problems using data; after the workshop, this increased to around 77%.



(a) Results of Q8: I feel confident in my ability to extract insights from data (identify patterns, identify anomalies, calculate predictions)



(b) Results of Q14: I believe using data science would enhance my apps and bring a positive impact on my community or the world.

Fig. 3. The most interesting results of the **pre-** and **post-**survey questions evaluating students' perception of data science and their skills to use data science to solve impactful real-world problem (on a Likert scale)

In summary, the students grasped the significance of data cleanup, recognizing anomalies as valuable insights crucial for addressing identified challenges. Engaging with these challenges fueled their enthusiasm for skill acquisition, with tangible outcomes instilling a sense of accomplishment. Witnessing their apps' real-world impact on societal issues profoundly transformed their perspectives, empowering them to embrace the potential of AI.

## VII. CONCLUSION

In conclusion, the work described in this paper is a promising start for Data Action, empowering students to engage in meaningful, hands-on, data-driven projects that address real-world challenges. The paper presents a novel approach to addressing the challenges faced by software engineering education in the era of AI and data-driven technologies. Recognizing the increasing importance of data literacy and the integration of ML and AI into software applications, we propose a shift towards a curriculum that emphasizes real-world problem-solving using data science techniques. By

advocating for cultivating a "data thinking" mindset among software engineering students, we aim to bridge the gap between traditional computational skills and the data-driven industry's evolving demands.

## ACKNOWLEDGMENT

Evan Patton (Lead App Inventor Developer), Selim Tezel (Education Team Lead), Robert Parks (Curriculum Developer), Teacher Lisa Miller and her students of Medford Vocational Technical High School (Research Study Participants), Jennet Zamanova (Student & Developer of the Example App - Diabetes Logbook)

## REFERENCES

- [1] T. H. Davenport and D. Patil. "Data scientist." In: Harvard business review 90.5 (2012), pp. 70–76.
- [2] K. Mike, N. Ragonis, R. B. Rosenberg-Kima, and O. Hazzan. "Computational thinking in the era of data science." In: Communications of the ACM 65.8 (2022), pp. 33–35.
- [3] Tedre, M., Denning, P., Toivonen, T. (2021, November). CT 2.0. In Proceedings of the 21st Koli Calling International Conference on Computing Education Research (pp. 1-8).
- [4] M. Frank and J. Walker. "Some key challenges for data literacy." In: The Journal of Community Informatics 12.3 (2016).
- [5] R. Acuña. "Developing a Data Science Course to Support Software Engineering Students". In 2023 IEEE 35th International Conference on Software Engineering Education and Training (CSEET) (pp. 127-131). IEEE. 2023.
- [6] J. M. Wing. "The data life cycle." In: Harvard Data Science Review 1.1 (2019), p. 6.
- [7] P. Li, X. Rao, J. Blase, Y. Zhang, X. Chu, and C. Zhang. "CleanML: A study for evaluating the impact of data cleaning on ml classification tasks." In: 2021 IEEE 37th International Conference on Data Engineering (ICDE). IEEE. 2021, pp. 13–24.
- [8] R. Bhargava and C. D'Ignazio. "Designing tools and activities for data literacy learners." In: Workshop on data literacy, Webscience. 2015.
- [9] R. Williams. "How to train your robot: project-based ai and ethics education for middle school classrooms." In: Proceedings of the 52nd ACM Technical Symposium on Computer Science Education. 2021, pp. 1382–1382.
- [10] A. Grillenberger and R. Romeike. "Developing a theoretically founded data literacy competency model." In: Proceedings of the 13th Workshop in Primary and Secondary Computing Education. 2018, pp. 1–10.
- [11] M. Tissenbaum, M. A. Sherman, J. Sheldon, and H. Abelson. "From computational thinking to computational action: Understanding changes in computational identity through app inventor and the internet of things." In: International Society of the Learning Sciences, Inc.[ISLS]., 2018.
- [12] A. Patcha and J.-M. Park. "An overview of anomaly detection techniques: Existing solutions and latest technological trends." In: Computer networks 51.12 (2007), pp. 3448–3470.
- [13] B. Plale and I. Kouper. "The centrality of data: data lifecycle and data pipelines." In: Data analytics for intelligent transportation systems. Elsevier, 2017, pp. 91–111.
- [14] Elhashemy, H., Parks, R., Kim, D. Y., Patton, E., Abelson, H. Empowering Learners with a Low-Barrier Mobile Data Science Toolkit.
- [15] Schwaber, K. (1997). Scrum development process. In Business Object Design and Implementation: OOPSLA'95 Workshop Proceedings 16 October 1995, Austin, Texas (pp. 117-134). Springer London.
- [16] J. Calzada Prado and M. Á. Marzal. "Incorporating data literacy into information literacy programs: Core competencies and contents." In: Libri 63.2 (2013), pp. 123–134.
- [17] Elhashemy, H. (2023). From Curious Kids to Data Wizards: An Empowerment Approach to Data Science Education.
- [18] Inventor, M. A., Explore, M. I. T. (2017). App inventor. [línea]. Disponible en: <http://appinventor.mit.edu/explore/>. [Accedido: 26-may-2015].



# A Survey Study on the State of the Art of Programming Exercise Generation using Large Language Models

1<sup>st</sup> Eduard Frankford  
Computer Science  
University of Innsbruck  
Innsbruck, Austria

eduard.frankford@uibk.ac.at

2<sup>nd</sup> Ingo Höhn  
Computer Science  
University of Innsbruck  
Innsbruck, Austria

ingo.hoehn@student.uibk.ac.at

3<sup>rd</sup> Clemens Sauerwein  
Computer Science  
University of Innsbruck  
Innsbruck, Austria

clemens.sauerwein@uibk.ac.at

4<sup>th</sup> Ruth Breu  
Computer Science  
University of Innsbruck  
Innsbruck, Austria

ruth.breu@uibk.ac.at

**Abstract**—This paper analyzes Large Language Models (LLMs) with regard to their programming exercise generation capabilities. Through a survey study, we defined the state of the art, extracted their strengths and weaknesses and finally proposed an evaluation matrix, helping researchers and educators to decide which LLM is the best fitting for the programming exercise generation use case. We also found that multiple LLMs are capable of producing useful programming exercises. Nevertheless, there exist challenges like the ease with which LLMs might solve exercises generated by LLMs. This paper contributes to the ongoing discourse on the integration of LLMs in education.

**Index Terms**—Programming Education, Programming Exercise Generation, Large Language Models, Artificial Intelligence, ChatGPT, Programming Exercise Generation Benchmark

## I. INTRODUCTION

The advent of the mainstream use of LLM based applications such as ChatGPT has marked a significant milestone in the field of artificial intelligence. State of the art LLMs have demonstrated remarkable capabilities in a variety of applications, ranging from feedback creation to code generation [1], [2]. In the realm of programming education, the potential of LLMs to automate and enhance the generation of learning materials, such as programming exercises, is a promising area of exploration. Despite a significant number of publications, a dedicated survey study that examines and compiles these first attempts at generating programming exercises is still missing.

This research seeks to address this gap by analyzing and synthesizing state-of-the-art literature about the application of LLMs in the field of programming exercise generation. This study is therefore guided by the following research questions:

- **RQ1:** What is the current state of the art in programming exercise generation using LLMs?

- **RQ2:** What are the strengths and weaknesses of existing solutions in programming exercise generation using LLMs?
- **RQ3:** How can different LLMs be evaluated regarding their suitability for exercise generation?

The significance of this research lies in its potential to provide a more complete view of the topic of exercise generation, helping educators and researchers gain a better understanding of how to optimize educational resources in programming education.

The structure of this paper is organized as follows: Section II offers an overview of selected related literature. Section III details the methodologies used. The core findings of the study are outlined in Section IV, with a discussion of the results provided in Section V. Key limitations of the study are addressed in Section VI, and the paper concludes with Section VII, encapsulating the primary insights and discussing the wider implications of the study.

## II. RELATED WORK

The utilization of LLMs across various domains, including software engineering and education in general, has been extensively documented, with significant contributions from Hou et al. [3], Hadi et al. [4], and Yan et al. [5]. Hou et al. categorize LLMs' applications in software engineering, highlighting their adaptability, a feature crucial for educational applications like programming exercise generation. Hadi et al. extend this perspective by exploring LLMs' broader implications, challenges, and architectures, emphasizing their transformative potential and ethical considerations. Yan et al. conducted a comprehensive review on the ethical and practical challenges of LLMs in education, revealing their versatility across various educational tasks, from content generation to feedback provision.

While these studies collectively describe the capabilities, applications, and challenges of LLMs in software engineering and education, there is still no survey study specifically addressing the generation and benchmarking of programming exercises using LLMs. This research addresses this gap by exploring the potential of LLMs to automate programming exercise generation and by developing an evaluation matrix to guide educators in selecting the most appropriate LLM for their needs.

### III. METHODOLOGICAL APPROACH

This study made use of the methodological approach of a survey study to offer a complete view of the state of the art, highlighting significant contributions, and to identify potential areas for future research. The approach encompasses the following key components.

#### A. Literature Collection Strategy

To conduct a comprehensive search, both traditional academic databases and Google Scholar were utilized due to the novelty of the topic. The traditional databases included: (1) ACM Digital Library, (2) IEEE Xplore, (3) ScienceDirect, and (4) SpringerLink. After searching the classical databases and Google Scholar, a round of snowballing was conducted [6].

#### B. Search Terms

The search terms were derived from the core topics of interest: LLMs and programming exercise generation. A combination of keywords related to “Large Language Models”, “Programming Exercise Generation” and “Educational Technology” was used to capture the relevant literature. The search was iteratively refined to include emerging terms and synonyms to ensure comprehensiveness.

#### C. Inclusion & Exclusion Criteria

In alignment with Garousi et al.’s [7] guidelines, this survey study adopts inclusion and exclusion criteria to ensure the relevance and timeliness of the literature reviewed. We focus on studies published from 2018 to 2023, reflecting the rapid developments in AI. We only selected studies that offer insights into LLM applications in programming exercise generation and are accessible in full text. Last but not least, only articles available in English were considered.

#### D. Stopping Criteria

The stopping criterion of data exhaustion was used. However, this survey study, examined the entire body of available literature for the defined search string. Given the limited scope of existing research, all relevant studies were covered before reaching a point of data saturation. This point is supported by the fact that even snowballing did not add new sources to the body of included literature.

## IV. RESULTS

This section presents the findings from the survey study, organized by research question.

#### A. RQ1: Current State in Programming Exercise Generation

Working implementations for programming exercise generation exist and produce sensible results [8]–[11]. The study by Sarsa et al. [8] offers the most detailed results, claiming LLMs are capable of generating sensible, novel, and readily applicable programming exercises. Of 120 investigated exercises, 75% were sensible, 81.8% were novel, and 76.7% had a matching sample solution. Denny et al. [10] reported similar results, which is expected since they used the same underlying model (Codex) and applied it to a similar task. Haluptzok et al. [11] generated programming exercises, not with the intention of providing them to humans, but rather for model training purposes. The purpose was to enhance the code generation capabilities of the tested model through the creation and solving of programming exercises.

One reoccurring prompting technique is the decomposition of exercise parts into: (1) problem statement, (2) template code, (3) solution and (4) test cases. For exercise generation, this allows to reduce the context size and increases the likelihood of generating more precise results [12]. Freitas et al. [9] used two different models, for different components of the exercise. For the problem statement, the Google T5 model was used, while for the generation of template code, Google CodeT5 was employed. According to the researchers, this demonstrated equally capable results to OpenAI’s GPT-3 model, while only using smaller open source models.

In general, programming exercise generation systems have been using various LLMs. Sarsa et al. and Denny et al. [8], [10] utilized Codex [13], which is based on the GPT-3 [14] architecture but fine-tuned specifically for programming tasks, originally powering the GitHub Copilot. Freitas et al. [9], in contrast, employed Google T5 and CodeT5 models. Haluptzok et al. [11] experimented with the open-source GPT-Neo [15] model in its various forms (125M, 1.3B, and 2.7B versions) along with Codex. GPT-Neo is similar to GPT-3 and was trained on the Pile dataset [16], which includes a substantial amount of GitHub code, providing a robust foundation for programming-related tasks.

#### B. RQ2: Strengths and Weaknesses of Existing Solutions

The main advantage of automated exercise generation lies in its remarkable ability to create learning material in a time efficient way. The traditional process of developing educational resources is labor-intensive and often demands a considerable degree of expertise. This efficiency opens up unprecedented possibilities for creating an extensive array of novel learning resources,

including detailed code explanations and comprehensive code examples, on a virtually limitless scale [17].

Haluptzok et al. [11] underscore the significant benefits of fine-tuning in the context of programming exercise generation. Specifically, fine-tuning Neo models on verified synthetic puzzle-solution pairs resulted in a two to five times improvement in puzzle-solving capabilities compared to the baseline model. This demonstrates a key strength of LLMs, a model with the general capacity of, for example GPT-4, when fine-tuned for programming exercise generation, could likely lead to impressive results.

Additionally, generated exercises, even if imperfect, can serve as a starting point for further modification by teachers or as a basis for student activities, like code reviewing and debugging exercises [8]. Similar observations are made by Kasneci et al. and Prather et al. [18], [19]. On the one hand, they emphasize personalizing the context of exercises, making them more engaging to the respective student. On the other hand, programming exercise generation enables more granular scaling by difficulty.

A weakness mentioned by Sarsa et al. [8] is the quality of the generated tests or test suites. Less than a third of exercises with tests pass successfully, and only about 70% of the exercises include tests at all. Therefore, the general precision of the generation method has still to be improved.

Another significant challenge, mentioned by Denny et al. [10], is that exercises generated using LLMs seem to be easily solvable by LLMs as well. This aspect introduces the risk of creating a counterproductive cycle where the exercises produced by LLMs fail to adequately challenge students. This is because students might find it tempting to simply input these exercises into a LLM-based application, such as ChatGPT, and obtain solutions with minimal effort.

Freitas et al. [9] mentioned a quality trade-off between costly commercial models like GPT-3, GPT-4 and freely accessible open source models, like Google T5 or GPT-Neo.

Other studies [17]–[19] highlight several disadvantages such as an over-reliance on the student as well as the teacher side, ethical concerns and licensing issues.

#### C. RQ3: Evaluating LLMs Suitability for Exercise Generation

Most benchmarks for LLMs evaluate their performance based on a set of problems that the models are expected to solve. The evaluation is then conducted by calculating the percentage of correctly solved problems [13], [14], [20], [21]. As the models are getting better, there has been the trend of changing benchmarks to contain more complex problems [20], [22]. Earlier benchmarks like the

*Winograd Schema Challenge* are being phased out since recent models neared human-level performance [20]. Improved benchmarks, like the *MMLU*, include harder and more specialized subjects [20].

In programming, the *HumanEval* metric, introduced in the Codex paper by OpenAI researchers [13], is a well-established benchmark [21]. A subsequent study comparing several LLMs' coding abilities also based its evaluation on *HumanEval* [23]. Liu et al. [24] proposed *EvalPlus*, which builds upon *HumanEval*, adding the ability to detect more incorrectly synthesized code.

During the survey study, we have found both manual and automated LLM assessment approaches [8]. Regarding manual assessment, Sarsa et al. [8] present four metrics: (1) Sensibleness, (2) Novelty, (3) Topicality and (4) Readiness for Use. *Sensibleness* is defined as whether the problem could be given to students to solve. *Novelty* is true if the programming exercise is not findable in Google or GitHub. *Readiness of use* considered the amount of manual work needed by a teacher before being able to use the generated problem. The concept, *topicality*, is related to *novelty* and examines how well a concept given in the prompt is accounted for in the created exercise.

The studies by Sarsa et al. and Denny et al. [8], [10] also reported five metrics that can be evaluated automatically: (1) Has sample solution, (2) Sample solution is executable, (3) Has tests, (4) All tests pass and (5) Test coverage.

The general drawbacks of a manual evaluation are subjectivity, bias and cost when compared to automated evaluations [13].

Having analyzed multiple approaches for benchmarking in general, it is important to derive a benchmark tailored to benchmarking programming exercise generation instead of general programming abilities of LLMs.

As a first step, it is important to define the key requirements of programming exercise generation. Only then, metrics can be established to measure model performance. Defining what constitutes a good exercise, which results in effective learning outcomes for students, is challenging. Typically, high-quality exercises share characteristics such as a well-defined problem statement, good structure, a specific competency focus and a progressively increasing difficulty for sub-tasks. Additionally, for the use in higher education, costs need to be taken into account, because decomposing the exercises and including previously generated exercise parts in the prompt, can lead to expensive requests when using paid models like GPT-4 [9]. The constructs of novelty and readiness of use have been proposed before and should also be taken into account to measure the quality of the generated exercises [8], [10]. Additionally,

TABLE I: Evaluation Matrix

	LLM
<b>General Assessment</b>	
Costs / Semester	Numeric
Data Privacy	Boolean
HumanEval Score	Percentage
PEG Benchmark	Percentage
<b>Program Analysis</b>	
Has sample solution	Percentage
Runnable sample solution	Percentage
Test Cases are present	Percentage
All tests pass	Percentage
Test coverage	Percentage
<b>Qualitative Assessment</b>	
Sensibility	Percentage
Novelty	Percentage
Readiness of Use	Percentage

a viable approach to establish a quantitative benchmark for programming exercise generation using LLMs could be a match-based multiple choice approach, similar to the *hellaSWAG* benchmark [25]. The *hellaSWAG* dataset evaluates the ability to complete unfinished sentences with multiple choice questions. Similarly, a dataset could be developed consisting of partial exercises. The model would then need to determine which of the provided answers best completes the exercise. This approach could be implemented as an automated benchmark, and named the *Programming Exercise Generation Benchmark (PEGB)*.

A more complete picture of LLM performance for programming exercise generation can be formed when evaluating a sample of generated exercises using the evaluation matrix presented in Table I. This takes into account general dimensions like cost, ability to generate code and data privacy, as well as specialized benchmarks like the before defined *PEGB* and constructs like sensibility, novelty and readiness of use. Using this matrix, educators may form a more evidence-based choice for a LLM to power their programming exercise generation service.

## V. DISCUSSION

The results presented in Section IV provide a starting point to better understand the current landscape and potential of generative AI in programming education, particularly in the context of programming exercise generation.

The current state of the art in programming exercise generation (RQ1), with Sarsa et al. [8], Freitas et al. [9], Denny et al. [10], and Haluptzok et al. [11], reveal a promising trend towards the practical application of LLMs to improve educational resource creation. These studies demonstrate the capability of LLMs to produce

novel, sensible, and ready-to-use programming exercises, which could significantly reduce the workload on educators and enable personalized learning experiences for students, which can be identified as their main strength. However, there are also significant weaknesses mentioned in the literature, such as the large failure quote of generated test suites and the potential for generated exercises to be easily solvable by LLMs themselves.

Last but not least, we found that there exist both automated and manual assessment approaches, which reflect the complexity of assessing AI-generated content's educational value.

## VI. LIMITATIONS

This study's comprehensiveness might be limited by the search strategy and terms used to gather relevant publications. Although a broad range of keywords and search techniques, including snowballing, were employed to ensure an extensive collection, there's still a possibility that some important studies were overlooked. Notably, the snowballing process did not reveal any new documents, suggesting the initial search was thorough.

Additionally, the initial screening and selection of studies were primarily conducted by a single individual, raising concerns about potential subjective bias in choosing which studies to include. This risk was partially mitigated through feedback from other researchers, yet the potential for bias cannot be fully discounted.

## VII. CONCLUSION

This study explored the use of LLMs for generating programming exercises, highlighting their potential to transform programming education. LLMs such as Codex and GPT-3 can produce engaging and novel exercises. However, they often require manual refinement to be classroom-ready. Despite this, using LLMs for exercise generation offers a substantial time saving possibility for educators and provides the flexibility to customize exercises to effectively meet student needs.

Nevertheless, over-reliance on LLMs poses risks to exercise quality and student learning outcomes, particularly as it seems like LLM-generated exercises are easily solvable by other LLMs.

To navigate the evolving landscape of LLMs in education, we proposed an evaluation matrix to provide a structured approach for assessing LLMs regarding their exercise generation capabilities. This matrix considers factors such as cost, data privacy, code generation capabilities, and the newly introduced concept of a programming exercise generation benchmark.

Looking ahead, completing the PEGB multiple choice programming exercise test set and regularly using the

evaluation matrix to assess emerging LLMs will be crucial to help educators and researchers to identify the most effective models for their needs.

## REFERENCES

- [1] E. Frankford, C. Sauerwein, P. Bassner, S. Krusche, and R. Breu, "Ai-tutoring in software engineering education," *arXiv preprint arXiv:2404.02548*, 2024.
- [2] P. Bassner, E. Frankford, and S. Krusche, "Iris: An ai-driven virtual tutor for computer science education," *arXiv preprint arXiv:2405.08008*, 2024.
- [3] X. Hou, Y. Zhao, Y. Liu, Z. Yang, K. Wang, L. Li, X. Luo, D. Lo, J. Grundy, and H. Wang, "Large language models for software engineering: A systematic literature review," *arXiv preprint arXiv:2308.10620*, 2023.
- [4] M. U. Hadi, R. Qureshi, A. Shah, M. Irfan, A. Zafar, M. B. Shaikh, N. Akhtar, J. Wu, S. Mirjalili *et al.*, "A survey on large language models: Applications, challenges, limitations, and practical usage," *Authorea Preprints*, 2023.
- [5] L. Yan, L. Sha, L. Zhao, Y. Li, R. Martinez-Maldonado, G. Chen, X. Li, Y. Jin, and D. Gašević, "Practical and ethical challenges of large language models in education: A systematic scoping review," *British Journal of Educational Technology*, vol. 55, no. 1, pp. 90–112, Aug. 2023.
- [6] C. Wohlin, "Guidelines for snowballing in systematic literature studies and a replication in software engineering," 2014.
- [7] V. Garousi, M. Felderer, and M. V. Mäntylä, "Guidelines for including grey literature and conducting multivocal literature reviews in software engineering," *Information and Software Technology*, vol. 106, 2019.
- [8] S. Sarsa, P. Denny, A. Hellas, and J. Leinonen, "Automatic generation of programming exercises and code explanations using large language models," *ICER 2022 - Proceedings of the 2022 ACM Conference on International Computing Education Research*, vol. 1, 2022.
- [9] T. C. Freitas, A. Costa Neto, M. J. a. V. Pereira, and P. R. Henriques, "NLP/AI Based Techniques for Programming Exercises Generation," in *4th International Computer Programming Education Conference (ICPEC 2023)*, ser. Open Access Series in Informatics (OASIcs), R. A. Peixoto de Queirós and M. P. Teixeira Pinto, Eds., vol. 112. Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023, pp. 9:1–9:12. [Online]. Available: <https://drops.dagstuhl.de/entities/document/10.4230/OASIcs.ICPEC.2023.9>
- [10] P. Denny, S. Sarsa, A. Hellas, and J. Leinonen, "Robosourcing educational resources - leveraging large language models for learnersourcing," in *CEUR Workshop Proceedings*, vol. 3410, 2022.
- [11] P. Haluptzok, M. Bowers, and A. T. Kalai, "Language models can teach themselves to program better," *arXiv*, 2023, 2207.14502.
- [12] J. Zamfirescu-Pereira, R. Y. Wong, B. Hartmann, and Q. Yang, "Why johnny can't prompt: How non-ai experts try (and fail) to design llm prompts," in *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, ser. CHI '23. New York, NY, USA: Association for Computing Machinery, 2023. [Online]. Available: <https://doi.org/10.1145/3544548.3581388>
- [13] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. de Oliveira Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, A. Ray, R. Puri, G. Krueger, M. Petrov, H. Khlaaf, G. Sastry, P. Mishkin, B. Chan, S. Gray, N. Ryder, M. Pavlov, A. Power, L. Kaiser, M. Bavarian, C. Winter, P. Tillet, F. P. Such, D. Cummings, M. Plappert, F. Chantzis, E. Barnes, A. Herbert-Voss, W. H. Guss, A. Nichol, A. Paino, N. Tezak, J. Tang, I. Babuschkin, S. Balaji, S. Jain, W. Saunders, C. Hesse, A. N. Carr, J. Leike, J. Achiam, V. Misra, E. Morikawa, A. Radford, M. Knight, M. Brundage, M. Murati, K. Mayer, P. Welinder, B. McGrew, D. Amodei, S. McCandlish, I. Sutskever, and W. Zaremba, "Evaluating large language models trained on code," *CoRR*, vol. abs/2107.03374, 2021. [Online]. Available: <https://arxiv.org/abs/2107.03374>
- [14] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," *CoRR*, vol. abs/2005.14165, 2020. [Online]. Available: <https://arxiv.org/abs/2005.14165>
- [15] S. Black, L. Gao, P. Wang, C. Leahy, and S. Biderman, "Gpt-neo: Large scale autoregressive language modeling with meshtensorflow," <https://doi.org/10.5281/zenodo.5551208>, 2021, accessed: 2024-01-02.
- [16] L. Gao, S. Biderman, S. Black, L. Golding, T. Hoppe, C. Foster, J. Phang, H. He, A. Thite, N. Nabeshima, S. Presser, and C. Leahy, "The pile: An 800gb dataset of diverse text for language modeling," *CoRR*, vol. abs/2101.00027, 2021. [Online]. Available: <https://arxiv.org/abs/2101.00027>
- [17] B. A. Becker, P. Denny, J. Finnie-Ansley, A. Luxton-Reilly, J. Prather, and E. A. Santos, "Programming is hard - or at least it used to be: Educational opportunities and challenges of ai code generation," in *SIGCSE 2023 - Proceedings of the 54th ACM Technical Symposium on Computer Science Education*, vol. 1, 2023.
- [18] E. Kasneci, K. Sessler, S. Küchemann, M. Bannert, D. Dementieva, F. Fischer, U. Gasser, G. Groh, S. Günemann, E. Hüllermeier, S. Krusche, G. Kutyniok, T. Michaeli, C. Nerdel, J. Pfeffer, O. Poquet, M. Sailer, A. Schmidt, T. Seidel, M. Stadler, J. Weller, J. Kuhn, and G. Kasneci, "Chatgpt for good? on opportunities and challenges of large language models for education," *Learning and Individual Differences*, vol. 103, 2023.
- [19] J. Prather, P. Denny, J. Leinonen, B. A. Becker, I. Alblawi, M. Craig, H. Keuning, N. Kiesler, T. Kohn, A. Luxton-Reilly, S. MacNeil, A. Peterson, R. Pettit, B. N. Reeves, and J. Savelka, "The robots are here: Navigating the generative ai revolution in computing education," *arXiv*, 2023, 2310.00658.
- [20] D. Hendrycks, C. Burns, S. Basart, A. Zou, M. Mazeika, D. Song, and J. Steinhardt, "Measuring massive multitask language understanding," *CoRR*, vol. abs/2009.03300, 2020. [Online]. Available: <https://arxiv.org/abs/2009.03300>
- [21] Z. Guo, R. Jin, C. Liu, Y. Huang, D. Shi, Supryadi, L. Yu, Y. Liu, J. Li, B. Xiong, and D. Xiong, "Evaluating large language models: A comprehensive survey," *arXiv*, 2023, 2310.19736.
- [22] A. Wang, Y. Pruksachatkun, N. Nangia, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman, "Superglue: A stickier benchmark for general-purpose language understanding systems," *CoRR*, vol. abs/1905.00537, 2019. [Online]. Available: <http://arxiv.org/abs/1905.00537>
- [23] F. F. Xu, U. Alon, G. Neubig, and V. J. Hellendoorn, "A systematic evaluation of large language models of code," in *Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming*, ser. MAPS 2022. New York, NY, USA: Association for Computing Machinery, 2022, p. 1–10. [Online]. Available: <https://doi.org/10.1145/3520312.3534862>
- [24] J. Liu, C. S. Xia, Y. Wang, and L. Zhang, "Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation," *arXiv*, 2023, 10.48550/ARXIV.2305.01210. [Online]. Available: <https://arxiv.org/abs/2305.01210>
- [25] R. Zellers, A. Holtzman, Y. Bisk, A. Farhadi, and Y. Choi, "Hellaswag: Can a machine really finish your sentence?" in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, 2019. [Online]. Available: <http://dx.doi.org/10.18653/v1/p19-1472>



# Leveraging Lecture Content for Improved Feedback: Explorations with GPT-4 and Retrieval Augmented Generation

1<sup>st</sup> Sven Jacobs

*Computer Science Education  
University of Siegen  
Siegen, Germany  
sven.jacobs@uni-siegen.de*

2<sup>nd</sup> Steffen Jaschke

*Computer Science Education  
University of Siegen  
Siegen, Germany  
steffen.jaschke@uni-siegen.de*

**Abstract**—This paper presents the use of Retrieval Augmented Generation (RAG) to improve the feedback generated by Large Language Models for programming tasks. For this purpose, corresponding lecture recordings were transcribed and made available to the Large Language Model GPT-4 as external knowledge source together with timestamps as metainformation by using RAG. The purpose of this is to prevent hallucinations and to enforce the use of the technical terms and phrases from the lecture. In an exercise platform developed to solve programming problems for an introductory programming lecture, students can request feedback on their solutions generated by GPT-4. For this task GPT-4 receives the students' code solution, the compiler output, the result of unit tests and the relevant passages from the lecture notes available through the use of RAG as additional context. The feedback generated by GPT-4 should guide students to solve problems independently and link to the lecture content, using the time stamps of the transcript as meta-information. In this way, the corresponding lecture videos can be viewed immediately at the corresponding positions. For the evaluation, students worked with the tool in a workshop and decided for each feedback whether it should be extended by RAG or not. First results based on a questionnaire and the collected usage data show that the use of RAG can improve feedback generation and is preferred by students in some situations. Due to the slower speed of feedback generation, the benefits are situation dependent.

**Index Terms**—Programming Education, Feedback, Large Language Models, GPT-4, Retrieval Augmented Generation

## I. INTRODUCTION

Individual support in teaching and learning contexts with heterogeneous learning groups is desirable in both school and university educational settings, but usually cannot be fully implemented in reality due to the limited availability of teaching staff. The topic of individual support by generative AI such as GPT-4 [1] is particularly promising in computer science education due to its good programming capabilities [2]. Commercially available applications such as ChatGPT, Bard, GitHub Copilot and others are not explicitly designed for skill development or knowledge acquisition, so they directly solve the given programming tasks instead of guiding the learner to solve the problem. This would only be possible with specific prompts. In addition, the external LLM application, such as ChatGPT, must be provided with the current code or error

messages each time. For this reason, we have developed the *Tutor Kai* programming exercise environment with integrated LLM support.

In order to support the student in exercises in the context of an associated lecture with teaching material with feedback, it seems useful if the feedback also refers to the corresponding content and is verifiably linked. This can be achieved by using Retrieval Augmented Generation (RAG), which can also reduce hallucinations because it is based more on real, verifiable facts [3]. The question then becomes how to design a feedback system that references and links to lecture information and how students perceive it.

## II. RELATED WORK

Even before LLMs were available, there were a variety of feedback systems for programming tasks [4]. While checking for correct syntax and semantics can be easily automated with unit tests, more specific feedback requires static code checks, which can be time-consuming to set up for each task [5].

The new possibilities of large language models are manifesting themselves in computer science education in several areas, such as the generation of teaching materials [6] and the analysis of student work [7]. The CodeAid system, for example, provides students with various programming aids such as Inline Code Exploration, Question from Code, Help Fix Code, Explain Code, and Help Write Code [8]. There are also solutions for automated feedback using LLMs [9] [10] [11] [12], although these do not include specific lecture information in the generated feedback.

While there are still no publications on knowledge-based feedback in programming education, videos are already widely used as a knowledge base for question answering [13] and chatbots [14]. Asthana et al. describe a system that uses lecture videos and transcripts as a knowledge base and extracts metadata from them using large language models, e.g. to generate questions about concepts. Feedback is also to be generated on this basis, but has not yet been evaluated [15].

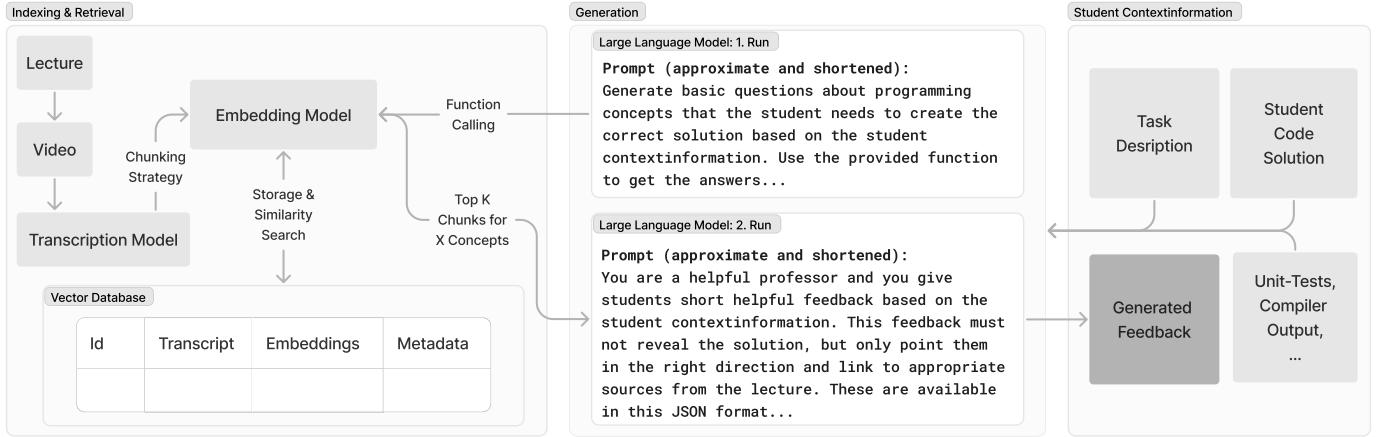


Fig. 1. System Design for Enhanced Programming Feedback using Retrieval Augmented Generation

### III. DESIGN AND IMPLEMENTATION

For lectures, the question arises as to which knowledge elements are suitable as a basis. For example, there are texts such as lecture slides, books, and worksheets combined with illustrations. However, the oral explanations of the teacher, which put the content into context, seem to be particularly suitable, since they are available in video form as a lecture recording. By linking the lecture recording to the corresponding timestamp, students are able to perceive the associated visual elements.

#### A. Indexing

The lecture recordings were first transcribed into .SRT (SubRip Text) format using the OpenAI speech recognition model Whisper [16]. This format not only provides the transcribed text, but also includes timestamps for each segment, allowing subsequent linking to specific points in the video. The texts assigned to a segment in the transcript can vary greatly in length, making them unsuitable for further processing. Therefore, a simple chunking strategy has been implemented, which reduces the text assigned to a segment to a uniform size of 512 characters with an overlap of 64 characters to the previous text. The associated timestamp is updated accordingly. A vector representation of the text is stored in a vector database using an embedding model, together with the original text and the start of the associated timestamp, as well as the name of the original video file. A Postgres database with the pgvector extension and text-embedding-ada-002 from OpenAI is used for Tutor Kai.

#### B. Retrieval

In naive RAG, the question is put into a vector representation as a query with the same embedding model for Question Answering, so that similarity scores between the query vector and the vectorized chunks within the indexed corpus can be computed [17]. The most similar chunks (top K) are then made available to the LLM for answer generation. However, in order to generate feedback on the solutions to programming tasks, there is no specific question or other text that would be suitable as a query for retrieval in the application. Therefore, a system

(Fig. 1) was implemented that first creates a suitable query for retrieval.

#### C. Generation

To implement the logic described, a prompt chain was implemented:

*1) First Run:* Similarly to the ReAct logic [18], the first run identifies X missing concepts for a correct solution based on the available student context information such as task description, student code solution, unit tests and compiler out. For each concept (e.g. recursion), the LLM formulates a simple question (e.g. "How does recursion work in Python?"), which is then used as a query in the described retrieval.

For Tutor Kai, this is implemented with GPT-4 (1106-preview and temperature = 0) and function calling. A maximum of 2 queries are generated for each feedback, for each of which the top 4 relevant chunks with associated meta information are retrieved. In total, a maximum of 8 relevant chunks with a length of 512 characters each are provided for the second run. In addition, the timestamps and video file names in the metainformation of the retrieved chunks are converted into a Markdown footnote format. In this way, a list

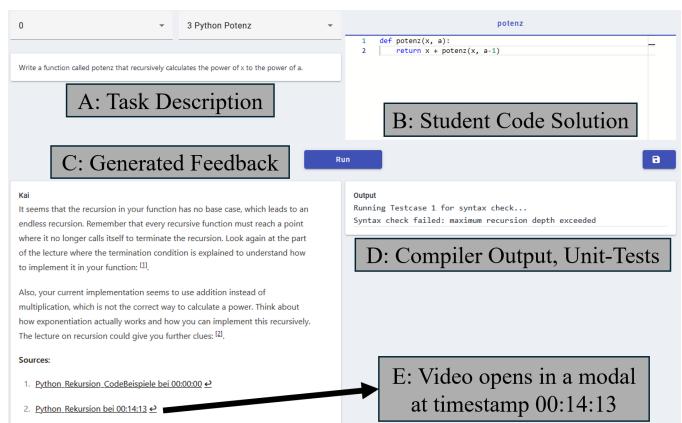


Fig. 2. User Interface of Tutor Kai (translated from German)

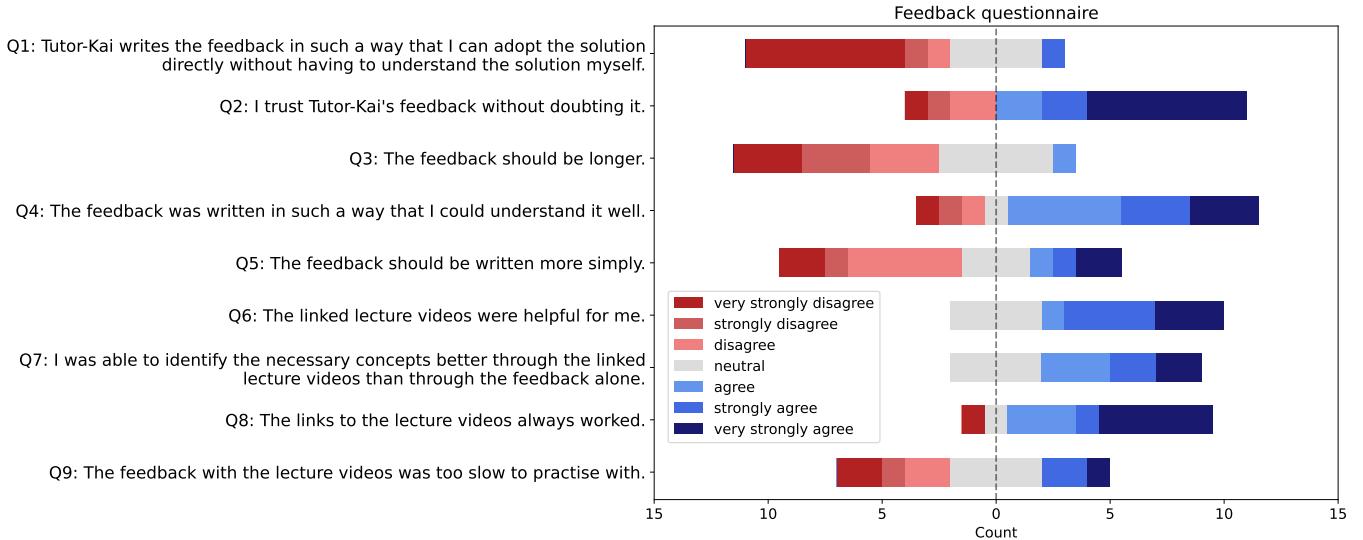


Fig. 3. Questionnaire result: Feedback (Questions translated from German)

of sources is automatically created in the final feedback by a markdown parser.

2) *Second Run:* In the second run, the final feedback is generated based on the retrieved chunks and the student context information (Fig. 2: A, B, D) using GPT-4 (1106-preview and temperature = 0). The prompt used includes the following elements:

- 1) Role description: The LLM is put in the role of a helpful professor.
- 2) Definition of rules: Outputting the solution and formulating the code is prohibited. Feedback should be no more than six sentences in no more than three paragraphs.
- 3) Description of the JSON format of the retrieved chunks with associated meta information.
- 4) Few Shot Examples on how to cite the retrieved chunks in the final feedback (using the provided markdown footnote link).
- 5) Student context information (task description, programming language, student code solution, compiler output, unit test result).
- 6) Retrieved lecture chunks in the described JSON format.

Using a markdown parser, linked lecture chunks appear as footnotes in feedback (Fig. 2: C). Based on the filename and timestamp, the videos can now be opened in a modal at the linked timestamp (Fig. 2: E).

#### IV. EVALUATION

As part of a voluntary exam preparation workshop, 15 students participated and used Tutor Kai for two to four hours in person and more online over the following three weeks. In addition to the familiar tasks from the previous semester, 10 new tasks were provided. Each time, students could choose between feedback with lecture information and feedback without lecture information. The feedback with lecture information was generated as described. The feedback without lecture

information does not use a prompt chain or retrieval, but only the 2nd run (Fig. 1) without the lecture information (same prompt without elements 3, 4 and 6).

After the workshop, the opinions of the students ( $n = 15$ ) about Tutor Kai and the generated feedback were surveyed using a questionnaire. It should be noted that not all students provided responses to every question posed. During and after the workshops, there were 2192 code submissions for which a total of 574 feedbacks were generated. Of these, 478 were feedback without lecture information and 96 were feedback with lecture information generated by the described system.

##### A. General Evaluation

An important goal of Tutor Kai is that the students do not receive knowledge of correct result in the feedback, but solve the problem independently, which is ensured by the prompt. This goal is achieved from the students' point of view (Fig. 3: Q1). Overall, the students are satisfied with both the simplicity and the length of the feedback and were able to understand it well (Fig. 3: Q3, Q4, Q5).

##### B. Comparing Feedback Types

In the 96 feedbacks with lecture information, 160 videosegments were linked (average = 1.67). These are spread across 57 different videosegments, with 3 specific segments being linked more than 10 times (maximum = 16).

Lecture information feedback is slower (time to the first streamed token) because the LLM response stream cannot begin until Run 1 (Fig. 1) is fully completed. The time to the first streamed token depends on the use of the OpenAI API, the length of the task description and the solution of the student's code. Therefore, it cannot be accurately predicted. In our tests, feedback with lecture content took about 18 seconds to stream the first token to the student, while feedback without lecture content took about 1 to 2 seconds.

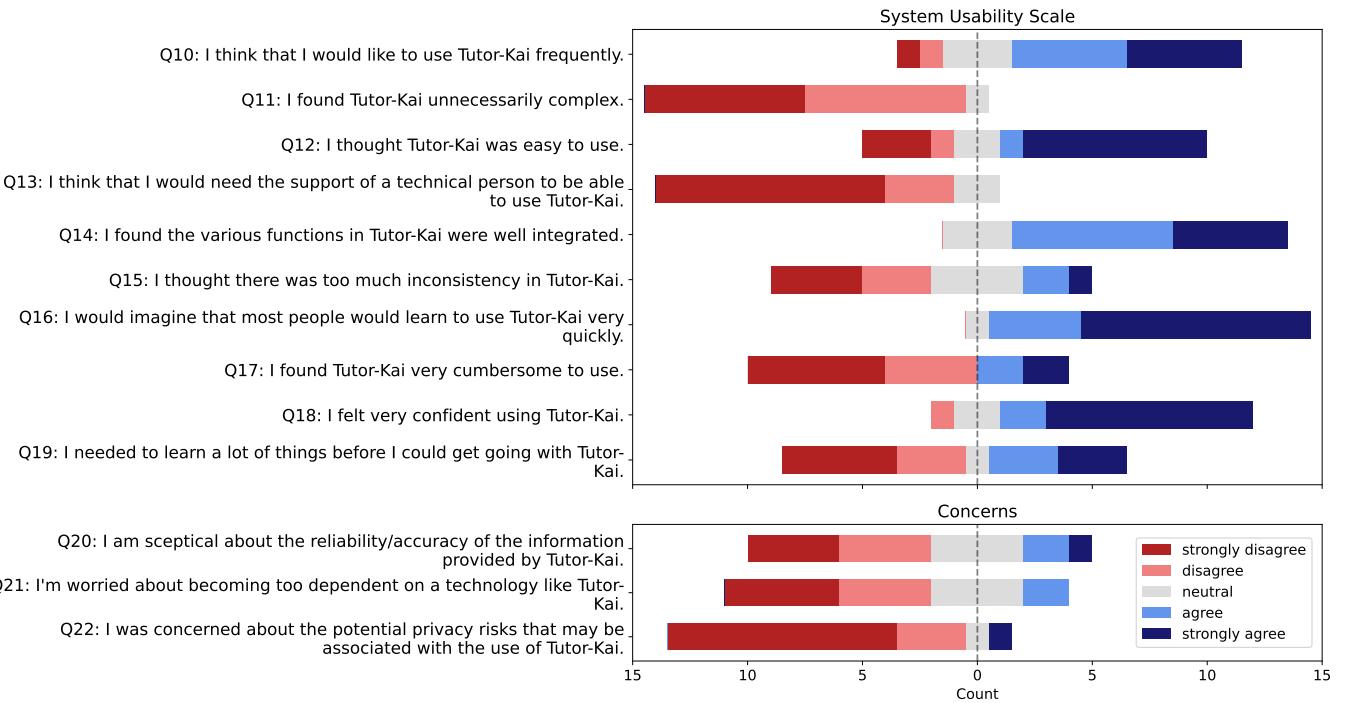


Fig. 4. Questionnaire result: System Usability and Concerns (Questions translated from German)

Students found the additional linking of lecture segments in the feedback helpful and it helped them find the necessary concepts to solve the problem (Fig. 3: Q6, Q7). However, some students found the generated feedback with lecture information too slow (Fig. 3: Q9). This perception is consistent with the additional responses collected in open-ended questions in the questionnaire about why they preferred which type of feedback. In this questionnaire, students mainly referred to the mentioned speed and several times described an approach where feedback without lecture content is generated first, which would be sufficient for "easy" cases, and feedback with lecture information is used for "more difficult" problems. In the open-ended questions it was mentioned multiple times that the feedback with lecture information helped to remember the lecture.

### C. Concerns

There are concerns that students will become too dependent on LLM-based support systems for programming [7]. In order to obtain preliminary results, three questions (Q20-Q22) from the TAME-ChatGPT [19] were included in the questionnaire. For the most part, students do not share this concern (Fig. 4: Q21). A possible reason for this is that, unlike ChatGPT and similar applications, Tutor Kai does not provide knowledge of the correct result.

The survey also showed that many students did not question the feedback even though they knew it was generated by generative AI (Fig. 3: Q2 and Fig. 4: Q20). When using such systems, a warning should therefore be displayed at all times.

### D. System Usability Scale

The system used was also evaluated for system usability using the System Usability Scale (Fig. 4) [20]. The final usability score was 74.8. Since the user interface consists of only a few clear elements, it should be straightforward to use. It should be noted that students rated their overall experience with Tutor Kai beyond the feedback, which also depends on the tasks to be completed. For example, Q19 (Fig. 4) may have been rated negatively by the students because they feel that they still have a lot to learn in order to complete the tasks successfully.

## V. LIMITATIONS AND FUTURE WORK

Due to the sample size of  $n=15$ , only trends are observable. Knowledge and skill acquisition were not the subject of this evaluation. To investigate the extent to which the linked videos were used, future studies should record how long each video was watched per feedback.

There is great potential for indexing and retrieval improvements for future work. For example, specially created short explanatory videos could be better suited than lecture recordings. Additional lecture content could also be linked. The current chunking strategy, which is simply based on the number of characters, could be improved by semantic chunking strategies or the use of a knowledge graph [21]. When generating the necessary concepts in the first run, this could be done several times and the most frequently selected ones could be used by majority voting. Solutions such as the feedback validation [12] will significantly improve the results in the future or enable completely new applications.

In this context, Nori et al. have shown that more advanced prompting techniques can lead to higher performance gains than the development of an improved foundation model [22].

## VI. CONCLUSION

This work investigates how to design a feedback system for programming tasks that refers to lecture content such as videos and can provide concrete content from this information. A two-run prompt chain is used, in which a query for retrieval augmented generation is generated in a first run with GPT-4. In a second run with GPT-4, the retrieved chunks from the transcribed lecture video are used together with a markdown footnote link containing the timestamp and name of the video. Together with the student's context information, the final feedback is generated, linking the corresponding lecture videos at the respective timestamp.

The system has been evaluated with 15 students. Most of them stated in a questionnaire that they found the feedback with linked lecture information helpful and that it helped them to find relevant concepts more quickly. The feedback without lecture information was preferred by students for quick feedback on what they considered a "simple" problem. The feedback with lecture information takes multiple times longer before the first token is streamed, because the query generation has to be completed first. This is another reason why feedback without lecture information was used about four times more often. The speed to the first character is therefore a trade-off. The students were also satisfied with the length and simplicity of the feedback.

## REFERENCES

- [1] OpenAI, "GPT-4 Technical Report," 2023, doi: 10.48550/ARXIV.2303.08774.
- [2] S. Bubeck, V. Chandrasekaran, R. Eldan, J. Gehrke, E. Horvitz, E. Kamar, P. Lee, Y. T. Lee, Y. Li, S. Lundberg, H. Nori, H. Palangi, M. T. Ribeiro, and Y. Zhang, "Sparks of Artificial General Intelligence: Early experiments with GPT-4," 2023, doi: 10.48550/arXiv.2303.12712.
- [3] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel, S. Riedel, and D. Kiela, "Retrieval-augmented generation for knowledge-intensive NLP tasks," in *Proceedings of the 34th International Conference on Neural Information Processing Systems*, ser. NIPS '20, NY, USA, 2020, doi: 10.48550/arXiv.2005.11401.
- [4] H. Keuning, J. Jeuring, and B. Heeren, "A Systematic Literature Review of Automated Feedback Generation for Programming Exercises," *ACM Transactions on Computing Education*, vol. 19, no. 1, pp. 1–43, 2019, doi: 10.1145/3231711.
- [5] J. Jeuring, H. Keuning, S. Marwan, D. Bouvier, C. Izu, N. Kiesler, T. Lehtinen, D. Lohr, A. Peterson, and S. Sarsa, "Towards Giving Timely Formative Feedback and Hints to Novice Programmers," in *Proceedings of the 2022 Working Group Reports on Innovation and Technology in Computer Science Education*, ser. ITiCSE-WGR '22. New York: ACM, 2022, pp. 95–115, doi: 10.1145/3571785.3574124.
- [6] S. Sarsa, P. Denny, A. Hellas, and J. Leinonen, "Automatic Generation of Programming Exercises and Code Explanations Using Large Language Models," in *Proceedings of the 2022 ACM Conference on International Computing Education Research - Volume 1*. Lugano and Virtual Event Switzerland: ACM, 2022, pp. 27–43, doi: 10.1145/3501385.3543957.
- [7] J. Prather, P. Denny, J. Leinonen, B. A. Becker, I. Albluwi, M. Craig, H. Keuning, N. Kiesler, T. Kohn, A. Luxton-Reilly, S. MacNeil, A. Petersen, R. Pettit, B. N. Reeves, and J. Savelka, "The Robots Are Here: Navigating the Generative AI Revolution in Computing Education," in *Proceedings of the 2023 Working Group Reports on Innovation and Technology in Computer Science Education*, ser. ITiCSE-WGR '23. New York: ACM, 2023, pp. 108–159, doi: 10.1145/3623762.3633499.
- [8] M. Kazemitaar, R. Ye, X. Wang, A. Z. Henley, P. Denny, M. Craig, and T. Grossman, "CodeAid: Evaluating a Classroom Deployment of an LLM-based Programming Assistant that Balances Student and Educator Needs," 2024, doi: 10.48550/arXiv.2401.11314.
- [9] N. Kiesler, D. Lohr, and H. Keuning, "Exploring the Potential of Large Language Models to Generate Formative Programming Feedback," in *2023 IEEE Frontiers in Education Conference (FIE)*. College Station, USA: IEEE, 2023, pp. 1–5, doi: 10.1109/FIE58773.2023.10343457.
- [10] A. Hellas, J. Leinonen, S. Sarsa, C. Koutchemi, L. Kujanpää, and J. Sorva, "Exploring the Responses of Large Language Models to Beginner Programmers' Help Requests," in *Proceedings of the 2023 ACM Conference on International Computing Education Research - Volume 1*, ser. ICER '23, vol. 1. New York: ACM, 2023, pp. 93–105, doi: 10.1145/3568813.3600139.
- [11] S. Jacobs and S. Jaschke, "Evaluating the Application of Large Language Models to Generate Feedback in Programming Education," in *2024 IEEE Global Engineering Education Conference (EDUCON)*. Kos, Greek: IEEE, 2024, doi: 10.48550/arXiv.2403.09744.
- [12] T. Phung, V.-A. Păducean, A. Singh, C. Brooks, J. Cambronero, S. Gulwani, A. Singla, and G. Soares, "Automating Human Tutor-Style Programming Feedback: Leveraging GPT-4 Tutor Model for Hint Generation and GPT-3.5 Student Model for Hint Validation," in *Proceedings of the 14th Learning Analytics and Knowledge Conference*. Kyoto, Japan: ACM, 2024, pp. 12–23, doi: 10.1145/3636555.3636846.
- [13] A. Madasu, J. Oliva, and G. Bertasius, "Learning to Retrieve Videos by Asking Questions," in *Proceedings of the 30th ACM International Conference on Multimedia*. Lisboa Portugal: ACM, 2022, pp. 356–365, doi: 10.1145/3503161.3548361.
- [14] M. Wölfel, B. S. Mehrnoush, A. Reich, and K. Anderer, "Knowledge-based and generative-ai-driven pedagogical conversational agents: A comparative study of grice's cooperative principles and trust," *Big Data and Cognitive Computing*, vol. 8, no. 1, pp. 1–20, 2024, doi: 10.3390/bdcc8010002.
- [15] S. Asthana, T. Arif, and K. C. Thompson, "Field experiences and reflections on using LLMs to generate comprehensive lecture metadata," *NeurIPS'23 Workshop on Generative AI for Education (GAIED)*, 2023.
- [16] A. Radford, J. W. Kim, T. Xu, G. Brockman, C. McLeavy, and I. Sutskever, "Robust Speech Recognition via Large-Scale Weak Supervision," in *Proceedings of the 40 Th International Conference on Machine Learning*, Honolulu, Hawaii, 2023, doi: 10.48550/arXiv.2212.04356.
- [17] Y. Gao, Y. Xiong, X. Gao, K. Jia, J. Pan, Y. Bi, Y. Dai, J. Sun, Q. Guo, M. Wang, and H. Wang, "Retrieval-Augmented Generation for Large Language Models: A Survey," Jan. 2024, doi: 10.48550/arXiv.2312.10997.
- [18] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao, "ReAct: Synergizing Reasoning and Acting in Language Models," in *International Conference on Learning Representations (ICLR)*, 2023, doi: 10.48550/arXiv.2210.03629.
- [19] M. Sallam, N. A. Salim, M. Barakat, K. Al-Mahzoum, A. B. Al-Tammemi, D. Malaeb, R. Hallit, and S. Hallit, "Assessing Health Students' Attitudes and Usage of ChatGPT," *JMIR Medical Education*, vol. 9, no. 1, pp. 1–15, 2023, doi: 10.2196/48254.
- [20] J. Sauro and J. R. Lewis, "Chapter 8 - Standardized usability questionnaires," in *Quantifying the User Experience (Second Edition)*, J. Sauro and J. R. Lewis, Eds. Boston: Morgan Kaufmann, 2016, pp. 185–248, doi: 10.1016/B978-0-12-802308-2.00008-4.
- [21] H. Abu-Rasheed, M. H. Abdulsalam, C. Weber, and M. Fathi, "Supporting Student Decisions on Learning Recommendations: An LLM-Based Chatbot with Knowledge Graph Contextualization for Conversational Explainability and Mentoring," in *Joint Proceedings of the 14th International Learning Analytics and Knowledge Conference (LAK24)*, Kyoto, Japan, 2024, doi: 10.48550/arXiv.2401.08517.
- [22] H. Nori, Y. T. Lee, S. Zhang, D. Carignan, R. Edgar, N. Fusi, N. King, J. Larson, Y. Li, W. Liu, R. Luo, S. M. McKinney, R. O. Ness, H. Poon, T. Qin, N. Usuyama, C. White, and E. Horvitz, "Can Generalist Foundation Models Outcompete Special-Purpose Tuning? Case Study in Medicine," 2023, doi: 10.48550/arXiv.2311.16452.



# Workplace Learning Ecology of Software Engineers and Implications for Teaching and Learning

Aditya Johri

Information Sciences &  
Technology  
College of Eng & Computing,  
George Mason University  
Fairfax, VA USA  
[johri@gmu.edu](mailto:johri@gmu.edu)

Aayushi Hingle Collier

ELAP, Linguistics and Comm  
Studies  
Montgomery College  
Montgomery, MD, USA  
[https://orcid.org/0000-0001-  
6350-4722](https://orcid.org/0000-0001-6350-4722)

Brent K. Jesiek

School of Engineering Education  
College of Engineering, Purdue  
University  
West Lafayette, IN USA  
[https://orcid.org/0000-0003-  
3056-5144](https://orcid.org/0000-0003-3056-5144)

Russell Korte

Department of Human and  
Organizational Learning  
The George Washington  
University  
Washington DC, USA  
[rkorte@gwu.edu](mailto:rkorte@gwu.edu)

Stephen Cory Brozina

Rayen School of Engineering  
Youngstown State University  
Youngstown, OH, USA  
[scbrozina@ysu.edu](mailto:scbrozina@ysu.edu)

**Abstract**—To prepare students for the future workforce, it is important to provide them with knowledge and guidance about professional software engineering practices. One critical element of workplace practices is learning on the job. There are few studies of how software engineers learn on the job and little focus within the literature on how this understanding can help train students. In this paper, we present a qualitative interview-based field study of workplace learning among professional software engineers. We conducted in-depth interviews with ten software engineers and analyzed the data using thematic analysis based on the *learning ecologies* framework. In our findings, we identify and discuss professionals' motivation and reasons for learning, the different resources they use to learn, the challenges they face in their learning, and their views on the relationship between formal education and on the job learning. We draw implications of the findings for teaching and learning, and future research.

**Keywords**—Workplace learning, field study, learning ecology, software engineering education

## I. INTRODUCTION

Technology workplaces are in a constant state of flux due to the increased speed of innovations. As work changes, so does the knowledge that is required to perform that work [1]. How do software engineers cope with the changing nature of their work? Specifically, how do they learn on the job and how can we better prepare students for the workplace? Software engineering education (SEE) already makes consistent efforts to bridge higher education and workforce through programs that provide students with apprenticeship opportunities [2]; by embedding the latest technical and professional communication knowledge and tools into the curriculum; and by providing industry-specific training [3]. Even if SEE cautiously embraces change and continues to prioritize theoretical and conceptual learning, it is important to keep up with practices of computing professionals to prepare the future workforce. This requires working with industry and implementing possible reforms [4].

Studies of software engineering and engineers have shown that learning on the job is one of the most important tasks

undertaken at work [5] but also argued [6] that a focus on workplace learning is still missing from the SEE literature. To understand how software engineers learn on the job, which is an important lifelong skill, we conducted an interview-based field study. Specifically, we used the framework of *learning ecologies* to interpretively understand why, how, and what software related learning on the jobs entails.

## II. LEARNING ECOLOGIES FRAMEWORK

The concept of 'ecology' originated in the natural sciences but is now commonly used in the social sciences to refer to a system of interdependent and mutually adaptive social and material relationships [7]. The metaphor of *learning ecology* – defined as an interdependent system of people, social relationships, and technology for learning – is useful for studying the workplace as it allows examination of the complexity of factors that shape workplace practices and impact learning, especially within high technology industries [7-8]. It also allows the inclusion of factors outside the organizational context that shape work and learning [8] and captures the nexus of learning where "educational and labor market institutions, colleges and workplaces, interact in the generation of learning ecologies that can be experienced as transformative, adaptive, or reproductive of social inequalities" [7, pg. 164]. That is, the learning ecologies framework recognizes that the workplace is not the only place of learning, there are also other social, spatial, and relational aspects that support the ecology. The other aspect of the framework is that it emphasizes that individuals can structure their environments for learning, creating affordances that were not there before and they can tactically pursue ends that extend beyond the immediate workplace [9]. Therefore, a worker who is motivated by self-interest or by their work can seize opportunities for learning through self-created learning spaces [8-9]. Overall, an ecological viewpoint helps in identifying factors related to not only what is learned but how and why [10], leading to the following **research question** in the context of software engineers in the workplace: What motivates learning, and what and how do they learn?

### III. RESEARCH STUDY

The study was designed as a qualitative field study with interviews as the primary data collection method [11-13]. The research study was approved by the Institutional Review Board of the researchers' university. Informed consent was acquired from all participants before data collection. The field study was conducted with an information technology company with around 180 employees with offices in a town in southern India. Participants ranged from those who were new to the organization to those who had been with the organization for years, including one of the co-founders. The interviews followed a semi-structured protocol. Participants worked on different projects across domains such as finance, social media, marketing, financial predictions, and human resource allocation, among others. The team sizes ranged from three to ten. Sampling was purposive and voluntary and after the company identified a team that was appropriate for the study in terms of size and range a of projects, a request was made of team members to voluntarily participate in the research study. There was a wide distribution of experience with both newcomers and those with 15+ years of experience included in the sample.

#### A. Data Collection and Analysis

In-person interviews ranging from 30-60 minutes were conducted with each of the ten participants using a semi-structured protocol. All interviews were recorded, transcribed, and analyzed using a three-step iterative coding process [14-15]. In the first step open coding of all ten transcripts was conducted and 341 codes identified. In the second round, codes were organized into 33 categories. The final stage of the coding process involved organizing the categories into themes that form the headings of the findings section. Throughout the process, at least two research team members coded the transcripts and compared codes to ensure reliability. The quotes have been edited for readability. Individual participants are identified in the quotes using "P" followed by a number (e.g., P1, P2, etc.).

## IV. FINDINGS

#### A. Purpose and Motivation for Learning on the Job

1) *Changes in the industry or profession:* The first motivation for learning on the job was the constantly changing nature of their industry, especially the advent of new platforms and technologies, and changes in tools or platforms being used:

*"I guess [change is] always there in the software industry... every six months or one year, there will be new concept, new topic to learn... To meet those requirements, we have to keep on learning. If the customer needs something and we don't know that, again, you have to learn that...if you want to survive, you have to learn, I guess." [P10]*

2) *Interest-Based Learning:* In addition to learning due to industry needs, participants also reported learning out of interest. As one interviewee explained:

*"Actually, some of the projects, we have done on our own thing. Things no client has asked, we have on our own – like there's a project called text summarization. Given a [data set], how you can summarize and showcase. Actually, this was not what any client asked. Out of our own interest, we*

*started reading the paper... But we have done that entirely of our own interest." [P2]*

3) *Learning the domain:* Participants worked on projects within specific domains such as finance, marketing, and education, and had to acquire domain knowledge. Depending on the project and the domain, they required different levels of expertise. Per two interviewees:

*"If I already know the [domain], then it will not take much time to understand the thing which I do. If that domain is entirely different, I have to put my 200% to know that domain before doing anything in it. Because, if you get into something, if you don't know anything about it, then sometimes it will make your life worse." [P6]*

*"It's a good idea [to learn the domain] as when analyzing the data, we may come up with a better solution, if we know the domain we are working on." [P10]*

In many cases, the clients were able to provide the engineers with enough training to be able to complete the work, but in others they had to develop knowledge beyond what the client told them. As one interviewee described it:

*"Also, that domain knowledge, it's not just domain knowledge; they should also have that experience of using that domain knowledge [in practice]...in our case, the client itself has that kind of experience...that's where the domain knowledge came from." [P4]*

4) *Technical Knowledge:* In addition to learning a domain, developing and updating technical skills was another primary motivation for participants to learn:

*"Even with platforms which you don't have to do much, you have to completely understand what they are trying to do...the new platforms... You just have to get behind it and try to see how it works." [P5]*

Yet the learning process was often challenging, as noted here:

*"Everything, whatever I work, I find it challenging. I try to do like, if I give solution A, it should work in all test cases.... If you want to cover all edge cases, though it is a very simple thing, it is challenging." [P9]*

*"The thing is, actually, if you think learning is a pressure then it's a pressure. If you think learning is a pleasure, then it's a pleasure...But pressure comes whenever there is faster implementation. Just learning is – always a pleasure, learning new things. But at the end [for a client]...you need to focus and get it done." [P2]*

#### B. Learning Resources and Process

1) *People as resources:* Other team members, family members, and friend networks were all resources that participants used to learn. Some of this was directed learning, to help solve a problem, but often it was an incidental or spontaneous part of other conversations:

*"Most of my friends – even my husband – are in the software industries, in the same field...yesterday we implemented something new...So, I ran home and shared this...When we have a conversation, we try to extend knowledge like that. If*

*"you meet my friend, if I meet my college friends, we will tell you what we are doing." [P6]*

Yet not everyone went to other people when they first faced a problem. Instead, they first tried to find solutions on their own, online. Asking others for help was also constrained by the type of projects, as some projects dealt with proprietary technology or knowledge and the discussion had to remain within the team:

*"More than friends, we rely on online [information]. Only when it won't help, do we go to friends. But normally, your friends would be almost at the same level." [P2]*

*"Actually, we will not go outside the team [because of proprietary issues], because each and every project will have a team lead, and managers and seniors." [P6]*

**2) Google or Searching:** Searching and using Google was mentioned as the first stop on the way to problem-solving by many participants. Search is the default interface for reaching many sources of information, including official documentation:

*"Any new things? We just search on Google; we try to find out the stuff that's understandable for us. If not, there are courses out there. So, Coursera and all those things. So, we try to get knowledge from there as well." [P9]*

*"I prefer to go for the documentation first. Visit their website, go to the documentation, and learn the basic things." [P10]*

*"If we're stuck in a problem, first we try to solve that. If it is not happening, we Google it. If we're not getting anywhere, we will post that question." [P6]*

**3) Blogs:** Blogs were mentioned by participants as one of their primary mechanisms for learning. Blog posts by other experts provided both an introduction to a topic and in-depth information needed to implement a solution. The text-based medium also meant that code could easily be shared.

*"Medium is good, actually, for when it comes to this thing. Medium is good. For Java, Bluebird so something, is there, so that is also good. Their blogs are good, actually." [P6]*

*"Rather than going for YouTube more, I go for the blogs and try to identify the similar problem statement. For example, if there's confusion with algorithms, how we can apply them, what type of problems we can typically apply, if we just go for the blog and look for that specific algorithm, we'll try to find for which type of classes we can apply this one." [P9]*

**4) Stack Overflow:** Another common resource mentioned by multiple interviewees was Stack Overflow:

*"Most of the time, I have copied or followed the Stack Overflow answer. Because that implementation might be much better than mine." [P3]*

*"Whenever I face a problem...I go directly to Stack Overflow first. If there is no information... then I go to documentation, where I get the details about the APIs." [P10]*

*"I will follow some specific users [on Stack]. I know a couple I can blindly use them because they've already proved themselves in the market." [P3]*

**5) Trying things out to learn:** In terms of the process, a final aspect that was mentioned was trying out different things:

*"Whenever I want to learn the new things, I first directly go and work hands on to do that... I get the better knowledge instead of only completely reading the documents, blogs and all." [P10]*

*"It's a trial-and-error method, basically. You cannot get a solution in a single go. Each step, expertise will help you, there are many people who have tons of expertise." [P5]*

Participants also mentioned that their needs in terms of resources changed over time as they gained more experience:

*"Tutorials, earlier, I used to watch when I was new, now I prefer documentation." [P10]*

**6) Others:** In addition to the above mentioned resources, participants also mentioned using YouTube, Coursera, Udemy, Kaggle, and other resources. For example, as one noted:

*"So first, I try to understand for myself, if I do not get it, then I will go to my seniors or go on YouTube. It will be like that. And then get some courses, like now we're doing web courses on Udemy.com." [P6]*

**7) Challenges to Learning:** Participants mentioned facing several challenges or barriers to their learning. First, when working on proprietary projects for clients, they were limited in what and how much they could share with others. This implied that other than the team working on the project, they could not reach out to others to seek help. As one participant noted:

*"Problem... basically, currently, it's in teams. The team itself, because as every project is confidential, we are not supposed to ask – or discuss the details. Currently, it's all within the team. Until and unless we can find different person in this organization, that's fair. Until then, we are not discussing anywhere else." [P5]*

It was also challenging for some of them to work on projects that required significant domain knowledge as it required both time and resources to learn a new domain:

*"[If] you really want to go for the domain knowledge, it's somewhat challenging. For finance, the financial, forex and all those things, the stuff which we get knowledge only after 5-6 years of expertise in that. So otherwise, we have some mentor, the guy that expertise in that." [P9]*

Limited access to infrastructure could become another barrier to learning. Some participants reported that they could try things out at a small scale but were unable to experiment at a larger scale, unsure of how things might work in production:

*"The basic features I can learn, whereas the scope of an industry level or the scope to produce solution to customer, I need a lot of data or something to implement it." [P5]*

Finally, another factor compounding all the above was a lack of time for learning. As one interviewee described it:

*"But in order to learn a lot of things, yes, you need time. Maybe a fifteen day, something like that. That kind of stuff, it's a never-ending job" [P5]*

Time constraints are real in the workplace, often imposed by the client, and make both working and learning challenging:

*"Our knowledge, it was time-constrained, too, in that project. So that makes me very good, after doing it well. It makes me very, very happy, I learned something new which nobody knows in this area. So I felt very good, actually. That was a very challenging thing." [P6]*

Other client-related issues were described as well, and these are important to document as such challenges might be 'created' for students in projects but do not have the same significance they do in the workplace. As two examples from participants:

*"Obviously whenever a client asks [for something], it gets priority over our own ideas." [P2]*

*"[When you are new], you find time to do it but when you're into [client] projects, it's really hard to find time to do courses...If you want to learn some new techniques or frameworks which is not in the project, but you find it interesting, definitely spending time [after] office, that for sure is needed, or you won't be up for the industry" [P5]*

### C. Relationship with formal learning

Participants additionally reflected on the link between their formal education in a college or university and their workplace learning. Primarily, they mentioned that in the workplace they get to learn the practical aspects of applying knowledge they had only theoretically learned in school:

*"With data theoretical knowledge...we just learn the formulas and concepts and derivations and all those things. We don't have much exposure to how we can apply it in real-time situations." [P3]*

*"In undergrad studies, we only knew there is something called [X], conceptually we knew. We had a course in college, a machine learning course...But after coming to [company name], we came to know practical implementation – the flaws in that particular model" [P8]*

This dynamic was especially true of courses such as distributed systems where access to infrastructure was lacking, or the complexity of the workplace was not available:

*"[We] had learned distributed systems in our curriculum but never implemented them in graduation. So, first time, you can see how they're managing, how they're distributing jobs, how they're doing it." [P1]*

When asked to reflect on what they would advise students and faculty in terms of teaching and learning in the formal environment, the participants emphasized the need to know the basics well so that you can use that knowledge to learn new technologies or languages. As one participant stated:

*"I would say, instead of directly going into the hype, like big data or data science, I would ask them to learn the basic things first. Basic things in the sense, any one they're interested in, programming language, like C, Java,*

*anything...if you're good at basics, I would say we can learn any other new technology which comes every six months or one year." [P10]*

Participants also emphasized the need to learn by doing, trying things out, and having some knowledge of what was state of the art in the industry:

*"They were not explaining, like, if you tell the professor they're going to give some lectures to students, according to me, as a professor, if I were a professor, after I explain this concept, I have to educate people like this concept right now in current industry, where it has been—' [P3]*

Finally, some participants reported going back to their university or college to give demos and to show students how things are done in the industry:

*"Sometimes, there comes a situation where my professor asks me to come to their college and tell students how you have actually worked on data engineering. That I have done many times. I used to go to those engineering colleges, used to get into a lab, show them the demos that I show." [P4]*

## V. DISCUSSION

**Learning Ecology:** The learning ecology of participants had multiple components. First, there was the motivation for learning, whether driven by a project or problem or self-directed. If self-directed, there was exploration that included doing online courses, trial-and-error with new data or systems or working with others on proof-of-concepts, among others. When directed by a client or for the sake of problem-solving, the learning was more targeted and would include using resources like Stack Overflow or asking a colleague. The external resources used by participants form vital support systems. Another crucial component of learning ecology involved understanding the work practices of the organization. Finally, the use of teams, meetings, SCRUM, Slack, and other ways to jointly work on a project are highly valuable resources and skills the participants needed to develop. Their participation in these practices was itself critical for learning about a domain, recent technologies, working with clients, and other skills essential for a successful career. Participants additionally acknowledged the importance of short courses on platforms such as Coursera and Udemy to learn about a new topic. Participants also referred to their college or university-level courses as important for their current work. Finally, the need for continuous learning across their career was another aspect of the ecology mentioned by participants [18].

**Implications for Teaching:** First, there is a need to encourage **informal learning** among students and for embedding online offerings such as MOOCs as part of students' education. Second, students need experience with Stack Overflow and similar **online platforms for problem-solving** and need to learn how to evaluate the responses on the platforms and how to deploy the solutions provided. Using GitHub and similar applications to both share their work and learn from others' outputs is another skill that needs to be integrated with programming and software engineering courses. Respondents also pointed out that learning the **basics, theoretical aspects** of computing and mathematics, are important skills and are ideally incorporated into the higher education curriculum. Finally, it is clear from the findings that certain aspects of the workplace are

hard and maybe **unnecessary to replicate** in higher education, such as working with clients on high-impact projects with real budgets and tight deadlines. Aspects of the projects can be incorporated in the curricula, but not the actual practice.

**Supporting Student Learning Ecology:** One way to implement findings from this work within higher education is through a *reflective exercise*. Reflection is a common teaching and assessment tool, and students can be asked to keep track of and map how they learn a specific topic, the tools they use, how they use them, and so on [16]. Another related approach is the use of *portfolios* for learning. Portfolios allow students to document and reflect over time, e.g., over a semester or even longer [17]. The advantage of these approaches is that students can be **more cognizant of their learning processes** and thereby build metacognition about how they learn best. Finally, providing direct exercises to students can guide them towards using specific resources and explaining how they did so. For instance, asking them to find an appropriate Stack Overflow response and then asking for an explanation. Similar exercises can be designed for other sources such as blogs [18].

**Implications for Future Research:** This study demonstrates that ‘workplace studies’ is a relevant and key area of research for SEE and research on learning and knowledge acquisition can help shed more light on how software engineers learn. Studies can further identify changes in the use of tools or processes within the workplace [19] and the cultural aspects of practices that shape learning [20].

**Implications for Use of Generative AI:** Although this study was conducted before the advent of GenAI in the workforce, the growing presence of GenAI tools as part of the broader learning ecology has several implications for higher education [21]. The integration of GenAI products such as Co-Pilot is necessary, and unavoidable, within the learning ecosystem but their use must be approached with some caution. As the study shows, tools and technologies change and a fundamental understanding of basic knowledge combined with an ability to learn new things is what makes for a successful career. A downside of GenAI tools is that rather than augmenting learning, they can become a substitute, and prevent students from developing skills for future learning.

## VI. CONCLUSION

Software engineering educators need to continue to integrate practical applications and tools within formal education. They also need to think broadly in terms of the overall learning ecology of students and how their course supports one or many elements of the ecology. By better understanding the contextual and cultural aspect of workplace learning, and a focus on how different aspects of learning come together, a learning environment can be created that provides students not only learn content knowledge but also lifelong learning skills.

## ACKNOWLEDGMENT

We thank the study participants and the U.S. National Science Foundation for funding this work.

## REFERENCES

- [1] M. Jazayeri, "The education of a software engineer," Proceedings. 19th International Conference on Automated Software Engineering, 2004., Linz, Austria, 2004.
- [2] M. Barr and S. W. Nabi, "The Development of Students' Employability Skills on a Work-Based Software Engineering Degree Programme," 2022 Proceedings of IEEE Frontiers in Education Conference (FIE), pp. 1-9.
- [3] Garousi, Vahid, Görkem Giray, Eray Tüzün, Cagatay Catal, and Michael Felderer. "Aligning software engineering education with industrial needs: A meta-analysis." Journal of Systems and Software 156 (2019): 65-83.
- [4] L. van der Duim, J. Andersson and M. Sinnema, "Good Practices for Educational Software Engineering Projects," 2007, Proceedings of ICSE, pp. 698-707.
- [5] Janice Singer, Timothy Lethbridge, Norman Vinson, and Nicolas Anquetil. 2010. An examination of software engineering work practices. In CASCON First Decade High Impact Papers (CASCON '10). IBM Corp., USA, 174–188.
- [6] D. Ford, T. Zimmermann, C. Bird and N. Nagappan, "Characterizing Software Engineering Work with Personas Based on Knowledge Worker Actions," Proceedings of 2017 ACM/IEEE ESEM, 2017, pp. 394-403
- [7] K. Evans. "Learning Ecologies at Work". In Barnett, R. & Jackson, N. (Eds.). Ecologies for Learning and Practice: Emerging Ideas, Sightings, and Possibilities. Routledge, 2019.
- [8] H-T. Le and A.Johri. "Engineers' situated use of digital resources to augment their workplace learning ecology." In 2021 IEEE Frontiers in Education Conference (FIE), pp. 1-8. IEEE, 2021.
- [9] N. Kersh and K. Evans. "Exploring working places and self-directed learning spaces at work". In A. Ostendorf & C. Permpoonwiwat (Eds.), Workplaces as learning spaces. Innsbruck, Australia: Innsbruck University Press, 2017.
- [10] S. W. Nabi, J. Maguire, S. Draper and Q. Cutts, "Keeping Software Engineering Students in Touch with Not Only What They are to Learn, But with Why," IEEE CSEE&T, Munich, Germany, 2020, pp. 1-5,
- [11] Josh Tenenberg. 2019. "Qualitative methods for computing education". In Cambridge handbook of computing education research, Sally Fincher and Anthony Robins (Eds.). Cambridge Univ. Press Cambridge, 173–207.
- [12] Lethbridge, Timothy C., Susan Elliott Sim, and Janice Singer. "Studying software engineers: Data collection techniques for software field studies." Empirical software engineering 10 (2005): 311-341.
- [13] E. H. Neto, F. M. C. van Amstel, F. V. Binder, S. dos S. Reinehr and A. Malucelli, "Trajectory and Traits of Devigners: A Qualitative Study about Transdisciplinarity in a Software Studio," 2020 CSEE&T, pp. 1-9.
- [14] A. L. Strauss. Qualitative analysis for social scientists. 1987. Cambridge university press.
- [15] H. Boeije. "A purposeful approach to the constant comparative method in the analysis of qualitative interviews" 2002. Quality and quantity 36, 391–409.
- [16] G. Glanville and B.A. Becker. "Reflective Learning Journals in Computer Science: The Student Experience". In The 9th International Conference on Engaging Pedagogy (ICEP 2016). ICEP, Dublin Ireland 2016 Dec 4.
- [17] H.L. Chen. "Using ePortfolios to support lifelong and lifewide learning". In Electronic Portfolios 2.0 2009 (pp. 29-35). Routledge.
- [18] A. Johri and S. Yang. "Scaffolded Help for Learning: How Experts Collaboratively Support Newcomer Participation in Online Communities." In Proceedings of the Communities and Technologies (C&T '17). ACM, New York, NY, USA, (2017): 149–158.
- [19] A. Johri. "Lifelong and lifewide learning for the perpetual development of expertise in engineering." European Journal of Engineering Education 47, no. 1 (2022): 70-84.
- [20] S. Secules, G. Pérez, R. Pea, and A. Johri. "Critical and Cultural Analysis of Engineering Learning." In Johri, A. (Ed.) International Handbook of Engineering Education Research, pp. 199-217. Routledge, 2023.
- [21] A. Johri, A. S. Katz, J. Qadir, and A. Hingle. "Generative artificial intelligence and engineering education." Journal of Engineering Education 112, no. 3 (2023): 572-577.



# Who is that girl? Drawing a portrait of a female student in Software Engineering

Yekaterina Kovaleva<sup>†</sup>  
LUT School of Engineering Science  
LUT University  
Lappeenranta, Finland  
yekaterina.kovaleva@lut.fi

Praveena Nair Sivasankaran  
Faculty of Innovation & Technology  
Taylor's University  
Subang Jaya, Malaysia  
praveena.nair@taylors.edu.my

Ari Happonen  
LUT School of Engineering Science  
LUT University  
Lappeenranta, Finland  
ari.happonen@lut.fi

**Abstract**— despite gender equality progress, women are underrepresented in computer science (CS) and software engineering (SE) fields due to entrenched stereotypes and limited exposure during early education. Educational institutions can address this by implementing gender-inclusive activities and enhancing female enrollment, environment, and learning processes. Understanding the traits and needs of female students is crucial for effective interventions. This scoping review of 53 studies from 2015 to 2022 reveals key findings such as high academic achievements, familial support, low programming experience, biases, and a desire to prove competence. Interestingly, despite self-doubt, women excel in project management within teams. The study offers insights for initiatives to rectify gender imbalances in CS and SE.

**Keywords**— women, gender equality, gender diversity, female, software engineering, computer science, stem, education, literature review

## I. INTRODUCTION

Despite notable progress in advancing gender equality, the participation of women in tech fields such as computer science (CS) and software engineering (SE) remains persistently low on a global scale, both within the educational realm and the industry [1], [2]. Increasing the number of female graduates in SE and CS is seen as a potential catalyst for positive societal changes [3]. Certain universities have initiated measures to attract more female students to predominantly male-dominated programs at various levels, including university-wide, program-specific, and course-specific initiatives [4], [5]. To design these measures, it is essential to understand female students' needs and characteristics. Consequently, this study seeks to construct a comprehensive understanding of the existing knowledge regarding women in CS and SE, as documented in the academic literature, and to identify potential avenues for further research. To fulfill this goal, the following research questions are utilized:

RQ1: What insights does the academic literature offer regarding female students in CS and SE?

RQ2: What categories of characteristics of female students emerge from the available data?

RQ3: Based on the analysis of this data, what potential directions could future research explore?

In following our Scoping Literature Review summarizes current knowledge on female students in SE and CS, discusses possible implications, and identifies ideas for future research.

## II. BACKGROUND

Despite progress towards gender equality, women's persistence in tech remains low [1]. The fields related to computing, such as CS and SE, have gender disparity in

education and industry. The researchers explain the low female persistence with social factors such as stereotypes, social norms, and habits [8], [9]. While women who study computing demonstrate a strong interest in the field [6], [10], the absence of early exposure [11] and the burden of stereotypes [8], [12] often dissuade them from pursuing careers in CS or SE. Due to societal pressures, girls frequently gravitate toward traditionally female-dominated fields [13] when selecting their educational paths [14]. As a result, the existing ratio of men and women in engineering is maintained, which produces a constant lack of female role models [15]. When there are no role models, resisting stereotypes becomes harder [16]. This can lead to new stereotypes, further strengthening harmful social norms that influence women's career decisions. At the same time, an increased number of female role models can disprove gender stereotypes. Therefore, it can be inferred that a growing cohort of female graduates in SE and CS may join the workforce, contributing to the amplification of positive societal changes provided that no other impediments hinder this progress [17].

Some universities worldwide are implementing different measures to attract more female students to male-dominant programs. For example, Payton et al. [18] and Narayanan et al. [19] put an effort into university community building. CS programs are looking for better ways to organize programming classes [20]. Some universities organize camps and hackathons to engage women in computing through informal educational experiences [21], [22]. These actions can be implemented effectively only by understanding the female students' characteristics and needs. This study was initiated to draw existing understanding from academic literature and identify directions for future research.

## III. METHOD

Scoping Literature Review was used to build an understanding of existing knowledge. The SLR attempts to understand the existing research activities comprehensively [23]. The search was performed systematically, and the sample is based on the search results from five academic publication databases (ACM, IEEE, Scopus, Web of Science, Science Direct). It covers the literature presenting gender-related observations in SE and CS tertiary education. These databases were chosen for great coverage of literature on the topic of interest. The study collected and analyzed the characteristics of women in SE and CS education.

The review process started with literature selection, which consisted of the following stages: keyword generation and tests, literature collection, and inclusion [24]. Firstly, study-

related keywords were generated and selected. The initial goal of the search was to collect all relevant literature concerning gender issues in SE and CS tertiary education. The final keywords list was formed after several search tests and modifications, as presented in Figure 1.

As SE and CS industries are growing fast, the literature search for publications was limited to full papers in the English language, not being older than the 2015 publication. After excluding duplicates, the total number of found publications was 882 unique studies. The process is presented in Figure 1. Then, the selection and exclusion processes were initiated. This process consisted of the following steps: inclusion criteria identification, title-based evaluation, abstracts-based evaluation, and finally, full text-based evaluation. For the inclusion, we have identified the following criteria:

- The study is fully available.
- Focus on female experience/gender differences.
- The study must be strongly connected to higher education, either being conducted within tertiary education institutions or closely related to activities within these institutions.
- The focus should be on computing-related fields, specifically CS, SE, or ICT. While studies encompass various STEM disciplines, our research is confined to computing-related studies.
- The study must offer gender-specific insights. This paper aims to construct a comprehensive understanding of women in CS and SE based on existing literature, emphasizing gender-specific observations.

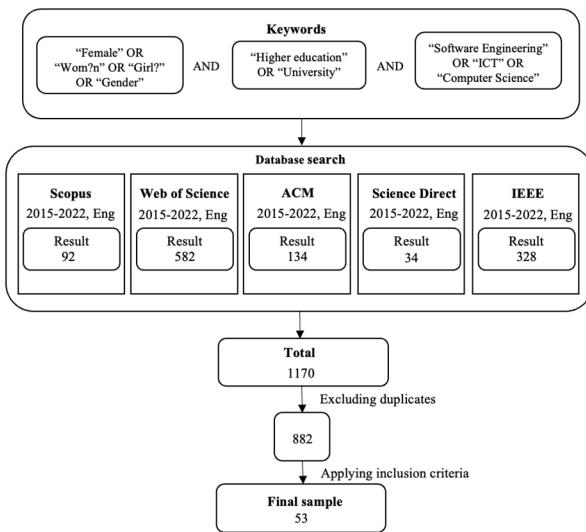


Fig. 1. Literature selection process.

During the evaluation, we selected 53 studies for further analysis. The following section contains a literature analysis of the selected studies.

#### IV. FINDINGS

For the literature analysis, we have adapted categories presented by Marklund and Gustavsson [25], who built the female in CS portrait based on the interviews. Their categories were: “motivation in CS studies,” “academic performance,” “motivation to persist in CS,” “experience with stereotypes,” and “experiences from the academic environment.” With some changes that better reflected the literature content and were grounded in the analyzed data, we ended up with the following categories of findings: background and experience, personal traits, motivation to study, factors influencing persistence, and academic experience.

##### A. Background and experience

Studies have identified common characteristics of female students in computing. These characteristics include high academic achievements [26], [27], [28], strong backgrounds in math and science [27], and parents who have careers in computing or engineering or are supportive of computing careers [27], [29], [30], [31]. However, female students are less likely to have programming experience or to have taken a high school computer science course [27], [32], [33]. Meanwhile, programming and other computer science experiences emerge as pivotal factors in shaping women's confidence in pursuing a career in computer science [27], [34], [35], [36]. Notably, women typically receive limited exposure to advanced computing knowledge at home and often encounter computing for the first time during their school education [37].

##### B. Personal traits

Studies show that female students undervalue their computing abilities compared to males [28], [32], [35], [38], [39], [40], [41]. Women in computer science even rate their math abilities lower than those in other STEM fields [28]. Self-evaluation in programming and self-efficacy in computer science are correlated for both genders, indicating that improved performance in programming tasks leads to enhanced efficacy in computer science and other computing skills [35]. Several studies showed that women are particularly interested in creativity [28], [29] and working for social good [34], [42], [43].

##### C. Motivation to study

Common motivations for studying computing include interest in the subject and career prospects [11], [25], [29], [37], [53]. Family members and educators also inspire young women [29], [30], [37], [41], [44]. When choosing colleges, female students extensively research and consult various sources, including current students and alumni [34]. An interesting theme in some studies [25], [41] was the desire to prove that women can do it.

##### D. Factors influencing persistence

Support from friends and family, particularly female friends, is a significant motivator for students to persist in academic programs [25], [30] [73]. This support network is crucial in class, as well as in female networks or external friendships. Additionally, observing peers' enthusiasm for computing inspires undecided students to pursue majors in the field [27].

### E. Academic experience

1) *Sense of Belonging.* Stereotypes in CS and SE, as outlined by Lewis et al. [16], often depict individuals as solely focused on CS, asocial, competitive, and male. However, many students feel they do not fit these stereotypes [16], [41], [45], [46]. Despite challenging these stereotypes, female CS students still face persistent biases [25], [41], [44], [47], [48], [49], [50] and often feel pressured to prove their competence to peers and educators, leading to the contemplation of leaving their studies or changing majors [25], [45], [49]. Lewis et al.'s research [16] showed that exposure to non-stereotypical examples can help students defy these expectations, highlighting the importance of diverse representation in education, industry, and media.

2) *Performance.* Although many studies indicate no difference in female and male academic performance, women seem to put extra effort into their courses [25] and feel the need for additional support [51].

3) *Teamwork.* Nguyen-Duc et al. [52] found gender-specific patterns in teamwork, noting that female students often took on project management and requirement-related tasks, with 70% leading project management and 56% handling requirements. They actively engaged in coordinating teams, planning, and tracking task execution while also participating in implementation (51%), user experience work (12%), and Scrum methodology management (9%). However, their involvement in architectural tasks was lower. While overall satisfaction with team dynamics was noted, it's unclear if women were assigned or volunteered for these roles. Kapoor et al. [49] discovered that positive leadership experiences prompt females to switch from a CS major or pursue a management career.

## V. DISCUSSION

In this study, we analyzed the main characteristics of female students in CS and SE. The findings are presented in Table 1.

TABLE I. FEMALE STUDENT IN CS AND SE PORTRAIT

Category	Characteristics
Background and experience	High academic performance
	Parents from CS or SE industry or supporting computing education
	Low programming and computing experience
	Exposure to computing happens at school
Personal traits	Low self-esteem in computing
	Interest in creativity
	Interest in social good
Motivation to study	Interest
	Career prospects
	Family inspirations
	Alumni inspirations
	Affirming the capabilities of women
Factors influencing persistence	Friends support and encouragement
	Family support
	Observing peer's enthusiasm
Academic experience	Low sense of belonging
	Experiencing biases
	Desire to drop out or switch the major
	Performance indicators are equal to men
	Having managerial position in a teamwork

Considering the findings, we would like to discuss the following implications of this study:

1) *Background:* the findings underscore the importance of early exposure to computing for females and the need to enhance their computing skills before university enrollment. Expanding recruitment beyond computing courses to identify undecided students not enrolled in CS classes is recommended. While strengthening the computing skills of the female population is very demanding and complex, there is an equally prospective solution – educating potential students, parents, and teachers about computing education and careers. This could be organized through school visits, online resources, outreach activities, etc. There is a chance that with active support, women could be more willing to pursue computing careers regardless of previous experience.

2) *Personal traits:* Educational organizations may adjust their programs to consider female interest in interdisciplinarity and creativity and emphasize the social significance of computing. The big issue that is hard to break through is self-doubt. While self-esteem levels of women in software engineering (SE) and computer science (CS) are frequently studied, research is scarce on the personal traits that positively impact women's pursuit and attainment of degrees in these fields. For example, the study of Townsend et al. [53] investigates the CS course experience from the perspective of female athletes. Female athletes' recognition of the significance of practice in sports mirrors the growth mindset evident in the computer science (CS) classroom, fostering resilience through experiences of wins and losses. This study offers a fresh outlook on the subject and proposes a solution for transitioning from self-doubt to enthusiasm.

3) *Motivation to study:* encouragement plays a crucial role in career choices, but women tend to receive less encouragement from parents and educators to pursue computer science (CS) than men [33]. Considering the strong influence of parents in choosing a career, the practical challenge for universities again would be to educate parents about CS and SE education opportunities and career prospects. An underexplored area is the desire of women to affirm their capabilities. It would be interesting to investigate whether this trait could be nurtured and evolve into a motivational framework for female students.

4) *Factors Influencing Persistence:* building social connections within the university setting is crucial. Various networking activities and events can offer additional opportunities for women to form friendships. These activities are an excellent opportunity to encourage active students to share their enthusiasm with peers. John and Carnoy's research also highlights the CS industry's specific requirements. It underscores that merely increasing the number of specialists does not guarantee employment if industry biases persist. Therefore, universities should enhance collaborations with industry [54] to advance global gender equality.

5) *Academic experience:* Universities and the teaching staff should become examples of zero-tolerance to discrimination and actively challenge existing stereotypes. One interesting theme that appears during the analysis is team composition. Future research may deepen understanding of women's roles in the team, whether they are assigned or

chosen voluntarily, and how educators can control this process.

## VI. CONCLUSION

This study presents the Scoping Literature Review of 53 papers observing gender-specific characteristics of students in CS, SE, and other ICT fields, covering background characteristics, personal traits, motivation to study and stay, and academic experiences. The study identified 20 attributes of female students and recommended implications based on these attributes.

Along with quite common recommendations, this paper specifically recommends educating parents and schoolteachers about computing education and careers, cultivating a growth mindset, affirming female capabilities, encouraging active female students to share their enthusiasm with peers, collaborating with industry, and promoting diversity in the workspace.

Future research could focus on identifying personal traits that encourage women to pursue SE or CS careers, delving into the dynamics of team composition, exploring whether women are assigned specific roles or choose them voluntarily, and how this knowledge can enhance gender diversity.

We recognize limitations in our study's scope, which focused exclusively on computer science (CS), software engineering (SE), and other information and communication technology (ICT) fields. Insights from other STEM areas were not included, suggesting potential avenues for future research to broaden understanding. Secondly, studies from different cultural environments have shown that evaluating the findings' applicability to different regions is essential. Thus, for instance, in Bangladesh, security concerns are one of the factors that need to be considered, among other factors [44]. In the Kingdom of Saudi Arabia (KSA), female CS students chose this country to study for an opportunity to work in a female-only environment, as 53% of women in KSA prefer to work without men in the workplace [55]. In India, one of the reasons families support a career in CS is marriage prospects [56]. Thus, while our study identifies common patterns, cultural contexts should be carefully considered when applying these insights.

## REFERENCES

- [1] A. H. Eagly, C. Nater, D. I. Miller, M. Kaufmann, and S. Sczesny, "Gender stereotypes have changed: A cross-temporal meta-analysis of U.S. public opinion polls from 1946 to 2018," *American Psychologist*, vol. 75, no. 3, pp. 301–315, Apr. 2020.
- [2] J. P. John and M. Carnoy, "The case of computer science education, employment, gender, and race/ethnicity in Silicon Valley, 1980–2015," *Journal of Education and Work*, vol. 32, no. 5, pp. 421–435, Jul. 2019.
- [3] H. de Ribaupierre, K. Jones, F. Loizides, and Y. Cherdantseva, "Towards Gender Equality in Software Engineering: The NSA Approach," in *2018 IEEE/ACM 1st International Workshop on Gender Equality in Software Engineering (GE)*, 2018, pp. 10–13.
- [4] J. L. Burnette, C. L. Hoyt, V. M. Russell, B. Lawson, C. S. Dweck, and E. Finkel, "A Growth Mind-Set Intervention Improves Interest but Not Academic Performance in the Field of Computer Science," *Social Psychological and Personality Science*, vol. 11, no. 1, pp. 107–116, Jan. 2020.
- [5] Y. Kovaleva, A. Happonen, and E. Kindsiko, "Designing gender-neutral software engineering program. stereotypes, social pressure, and current attitudes based on recent studies," in *Proceedings of the Third Workshop on Gender Equality, Diversity, and Inclusion in Software Engineering*, Pittsburgh Pennsylvania: ACM, May 2022, pp. 43–50.
- [6] R. Colomo-Palacios, N. B. Yahia, X. Larrucea, and C. Casado-Lumbreras, "Is the Gender Gap Narrowing in Higher Education Computing Studies? The Case of Norway, Spain, and Tunisia," *IEEE Revista Iberoamericana De Tecnologias Del Aprendizaje-IEEE RITA*, vol. 15, no. 4, pp. 336–343, Nov. 2020.
- [7] "Software developers: distribution by gender 2022 | Statista," Accessed: Oct. 12, 2023. [Online]. Available: <https://www.statista.com/statistics/1126823/worldwide-developer-gender/>
- [8] N. Ellemers, "Gender Stereotypes," no. Vol. 69, pp. 275–298, 2017, doi: 10.1146/annurev-psych-122216-011719.
- [9] M. Kovacic and C. E. Orso, "Historical roots of women's sorting into STEM occupations," *SSRN Journal*, 2024.
- [10] J. Wang and S. H. Moghadam, "Diversity Barriers in K-12 Computer Science Education: Structural and Social," in *Proceedings Of The 2017 ACM SIGCSE Technical Symposium on Computer Science Education (SIGCSE '17)*, Assoc Computing Machinery, 2017, pp. 615–620.
- [11] S. Kamberi, "Enticing Women to Computer Science with Es (Expose, Engage, Encourage, Empower)," in *2017 IEEE Women In Engineering (WIE) Forum USA East*, IEEE, 2017.
- [12] Y. Kovaleva, S. Hyrynsalmi, A. Saltan, A. Happonen, and J. Kasurinen, "Becoming an entrepreneur: A study of factors with women from the tech sector," *Information and Software Technology*, vol. 155, p. 107110, Mar. 2023.
- [13] R. Baltà-Salvador, M. Peña, A.-I. Renta-Davids, and N. Olmedo-Torre, "The intersection of sex and field: an examination of career choice factors and dropout intentions in STEM and non-STEM degrees," *European Journal of Engineering Education*, pp. 1–16, Feb. 2024.
- [14] L. Tsui, "Recruiting Females into Male Dominated Programs: Effective Strategies and Approaches," *Journal of College Admission*, 2009, Accessed: Sep. 22, 2021. [Online]. Available: <https://eric.ed.gov/?id=EJ838697>
- [15] B. J. Drury, J. O. Siy, and S. Cheryan, "When Do Female Role Models Benefit Women? The Importance of Differentiating Recruitment From Retention in STEM," *Psychological Inquiry*, vol. 22, no. 4, pp. 265–269, Oct. 2011.
- [16] C. M. Lewis, R. E. Anderson, and K. Yasuhara, "'I Don't Code All Day': Fitting in Computer Science When the Stereotypes Don't Fit," in *Proceedings of the 2016 ACM Conference on International Computing Education Research*, Melbourne VIC Australia: ACM, Aug. 2016, pp. 23–32.
- [17] E. Tereshchenko, A. Happonen, and V. Hasheela-Mufeti, "Barriers for Females to Pursue Stem Careers and Studies at Higher Education Institutions (HEI). A Closer Look at Academic Literature," *IJCSES*, vol. 14, no. 1/2/3/4, pp. 01–23, Aug. 2023.
- [18] J. Payton *et al.*, "STARS Computing Corps: Enhancing Engagement of Underrepresented Students and Building Community in Computing," *Comput. Sci. Eng.*, vol. 18, no. 3, pp. 44–57, May 2016.
- [19] S. Narayanan *et al.*, "Upward Mobility for Underrepresented Students: A Model for a Cohort-Based Bachelor's Degree in Computer Science," in *SIGCSE '18: Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, Assoc Computing Machinery, 2018, pp. 705–710.
- [20] S. K. Kuttal, K. Gerstner, and A. Bejarano, "Remote Pair Programming in Online CS Education: Investigating through a Gender Lens," in *2019 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, Oct. 2019, pp. 75–85.
- [21] Y. Kovaleva, A. Happonen, and A. Mbogho, "Towards gender balance in modern hackathons: literature-based approaches for female inclusiveness," in *Proceedings of the Third Workshop on Gender Equality, Diversity, and Inclusion in Software Engineering*, Pittsburgh Pennsylvania: ACM, May 2022, pp. 19–26.
- [22] J. Miller, S. Raghavachary, and A. Goodney, "Benefits of Exposing K-12 Students to Computer Science through Summer Camp Programs," in *2018 IEEE Frontiers in Education Conference (FIE)*, Oct. 2018, pp. 1–5.
- [23] G. Paré, M.-C. Trudel, M. Jaana, and S. Kitsiou, "Synthesizing information systems knowledge: A typology of literature reviews," *Information & Management*, vol. 52, no. 2, pp. 183–199, Mar. 2015.
- [24] K. Petersen, S. Vakkalanka, and L. Kuzniarz, "Guidelines for conducting systematic mapping studies in software engineering: An update," *Information and Software Technology*, vol. 64, pp. 1–18, Aug. 2015.
- [25] M. Marklund and S. Gustavsson, "'Why Am I Even Doing This?': The Experiences of Female Students in CS from an Insider Perspective," in *2018 International Conference on Learning and*

- Teaching in Computing and Engineering (LaTICE)*, Auckland, New Zealand: IEEE, Apr. 2018, pp. 77–81.]
- [26] L. Alonso-Virgós, M. D. Fondón, J. P. Espada, and R. G. CRESPO, “Women in science and technology studies. A study about the influence of parents on their children’s choice of speciality. And about the trend of the different specialities in Spanish students,” in *2021 IEEE Global Engineering Education Conference (EDUCON)*, 2021, pp. 122–130.
- [27] K. J. Lehman, “An Untapped Recruitment Pool: Undecided Students in CS1 Courses,” in *2019 Research on Equity and Sustained Participation in Engineering, Computing, and Technology (RESPECT)*, Feb. 2019, pp. 1–8.
- [28] K. J. Lehman, L. J. Sax, and H. B. Zimmerman, “Women planning to major in computer science: Who are they and what makes them unique?,” *Computer Science Education*, vol. 26, no. 4, pp. 277–298, 2016.
- [29] A. Funke, M. Berges, and P. Hubwieser, “Different Perceptions of Computer Science,” in *2016 International Conference on Learning and Teaching in Computing and Engineering (LaTICE)*, Apr. 2016, pp.
- [30] Y. Rankin, M. Agharazidermani, and J. Thomas, “The Role of Familial Influences in African American Women’s Persistence in Computing,” in *2020 Research on Equity and Sustained Participation in Engineering, Computing, and Technology (RESPECT)*, 2020, pp. 1–8.
- [31] T. Stanko and O. Zhiros, “Young women who choose IT: what role do their families play?,” in *2017 7th World Engineering Education Forum (WEEF)*, 2017, pp. 88–93.
- [32] N. Pirttinen, A. Hellas, L. Haaranen, and R. Duran, “Study Major, Gender, and Confidence Gap: Effects on Experience, Performance, and Self-Efficacy in Introductory Programming,” in *2020 IEEE Frontiers in Education Conference (FIE)*, Oct. 2020, pp. 1–7.
- [33] J. Wang and S. H. Moghadam, “Diversity Barriers in K-12 Computer Science Education: Structural and Social,” in *Proceedings of the 2017 ACM SIGCSE Technical Symposium On Computer Science Education (SIGCSE’17)*, Assoc Computing Machinery, 2017, pp. 615–620.
- [34] G. Archer, L. Bohmann, A. Carter, C. Cischke, L. M. Ott, and L. Ureel, “Understanding Similarities and Differences in Students Across First-Year Computing Majors,” in *2016 IEEE Frontiers in Education Conference (FIE)*, IEEE, 2016.
- [35] M. Kallia and S. Sentance, “Are boys more confident than girls? The role of calibration and students’ self-efficacy in programming tasks and computer science,” in *WIPSC’18: Proceedings of the 13th Workshop in Primary and Secondary Computing Education*, A. Muhling and Q. Cutts, Eds., Assoc Computing Machinery, 2018, pp. 87–90.
- [36] C. Milesi, L. Perez-Felkner, K. Brown, and B. Schneider, “Engagement, Persistence, and Gender in Computer Science: Results of a Smartphone ESM Study,” *Frontiers in Psychology*, vol. 8, Apr. 2017.
- [37] S. Smith, E. Sobolewska, J. Bhardwaj, and K. Fabian, “Exploring women’s motivations to study computer science,” in *2018 IEEE Frontiers In Education Conference (FIE)*, IEEE, 2018.
- [38] S. M. Hyrynsalmi and S. Hyrynsalmi, “Software Engineering Studies Attractiveness for the Highly Educated Women Planning to Change Career in Finland,” in *2019 IEEE/ACM 41ST International Conference on Software Engineering: Companion Proceedings (ICSE-COMPANION 2019)*, IEEE, 2019, pp. 304–305.
- [39] C. Schimpf, K. Andronicos, and J. Main, “Using life course theory to frame women and girls’ trajectories toward (or away) from computing: Pre high-school through college years,” in *2015 IEEE Frontiers in Education Conference (FIE)*, 2015, pp. 1–9.
- [40] C. Schindler and M. Mueller, “Gender Gap? A Snapshot of a Bachelor Computer Science Course at Graz University of Technology,” in *13th European Conference on Software Architecture (ECSA 2019)*, vol 2, L. Duchien, A. Koziolek, R. Mirandola, E. M. N. Martinez, C. Quinton, R. Scandariato, P. Scandurra, C. Trubiani, and D. Weyns, Eds., Assoc Computing Machinery, 2019, pp. 100–104.
- [41] J. Yates and A. C. Plagnol, “Female computer science students: A qualitative exploration of women’s experiences studying computer science at university in the UK,” *Education And Information Technologies*.
- [42] B. Ioannis and K. Maria, “Gender and student course preferences and course performance in Computer Science departments: A case study,” *Education and Information Technologies*, vol. 24, no. 2, pp. 1269–1291, Mar. 2019.
- [43] G. Potvin *et al.*, “Gendered Interests in Electrical, Computer, and Biomedical Engineering: Intersections With Career Outcome Expectations,” *IEEE Transactions on Education*, vol. 61, no. 4, pp. 298–304, Nov. 2018.
- [44] M. Tasmin, N. Ahmed, and T. Motahar, “Gender Disparity in Computer Science Education in Bangladesh: A Study of Women’s Participation in Computer Science,” in *2019 IEEE International Conference on Engineering, Technology and Education (TALE)*, 2019, pp. 1–7.
- [45] E. Hoehne and L. Zander, “Sources of Male and Female Students’ Belonging Uncertainty in the Computer Sciences,” *FRONTIERS IN PSYCHOLOGY*, vol. 10, Aug. 2019, doi: 10.3389/fpsyg.2019.01740.
- [46] C. Mooney, B. A. Becker, L. Salmon, and E. Mangina, “Computer Science Identity and Sense of Belonging: A Case study in Ireland,” in *2018 IEEE/ACM 1st International Workshop on Gender Equality in Software Engineering (GE 2018)*, IEEE, 2018, pp. 1–4.
- [47] A. Garcia-Holgado, M. Estrada, G. Marín-Raventós, and F. J. Garcia-Péñalvo, “Gender gap perception of computer science students in Costa Rica: A case study in two public universities,” in *CEUR Workshop Proceedings*, 2021, pp. 12–21.
- [48] N. Jaumot-Pascual, M. Ong, C. Silva, and A. Martinez-Gudapakkam, “Women of Color Leveraging Community Cultural Wealth to Persist in Computing and Tech Graduate Education: A Qualitative Meta-Synthesis,” *Education Sciences*, vol. 11, no. 12, Dec. 2021.
- [49] A. Kapoor and C. Gardner-McCune, “Considerations for Switching: Exploring Factors behind CS Students’ Desire to Leave a CS Major,” in *ITICSE’18: Proceedings of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education*, I. Polycarpou, J. C. Read, P. Andreou, and M. Armoni, Eds., ASSOC COMPUTING MACHINERY, 2018, pp. 290–295.
- [50] J. B. Main and C. Schimpf, “The Underrepresentation of Women in Computing Fields: A Synthesis of Literature Using a Life Course Perspective,” *IEEE Transactions on Education*, vol. 60, no. 4, pp. 296–304, 2017.
- [51] M. Menekse, X. Zheng, and S. Anwar, “Computer science students’ perceived needs for support and their academic performance by gender and residency an exploratory study,” *Journal of Applied Research in Higher Education*, vol. 12, no. 5, pp. 1025–1044, Feb. 2020.
- [52] A. Nguyen-Duc, L. Jaccheri, and P. Abrahamsson, “An Empirical Study on Female Participation in Software Project Courses,” in *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, 2019, pp. 240–241.
- [53] G. C. Townsend, K. J. Stewart, and S. Tunguz, “Tackling the Underrepresentation of Women in Computing and Finding Novel Help in Athletics,” in *2020 Research on Equity and Sustained Participation in Engineering, Computing, and Technology (RESPECT)*, 2020, pp. 1–4.
- [54] E. Tereshchenko, E. Salmela, E. Melkko, S. K. Phang, and A. Happonen, “Emerging best strategies and capabilities for university–industry cooperation: opportunities for MSMEs and universities to improve collaboration. A literature review 2000–2023,” *J Innov Entrep*, vol. 13, no. 1, p. 28, May 2024.
- [55] F. Alghamdi, “Why do female students choose to study CS in the Kingdom of Saudi Arabia?,” in *2017 5TH International Conference on Learning and Teaching in Computing And Engineering (LaTICE 2017)*, IEEE, 2017, pp. 49–53.
- [56] D. Thakkar, N. Sambasivan, P. Kulkarni, P. K. Sudarshan, and K. Toyama, “The Unexpected Entry and Exodus of Women in Computing and HCI in India,” in *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems (CHI 2018)*, Assoc Computing Machinery, 2018.



# Automated Programming Exercise Generation in the Era of Large Language Models

Niklas Meißner 

*Institute of Software Engineering  
University of Stuttgart  
Stuttgart, Germany*  
niklas.meissner@iste.uni-stuttgart.de

Sandro Speth 

*Institute of Software Engineering  
University of Stuttgart  
Stuttgart, Germany*  
sandro.speth@iste.uni-stuttgart.de

Steffen Becker 

*Institute of Software Engineering  
University of Stuttgart  
Stuttgart, Germany*  
steffen.becker@iste.uni-stuttgart.de

**Abstract**—Lecturers are increasingly attempting to use large language models (LLMs) to simplify and make the creation of exercises for students more efficient. Efforts are also being made to automate the exercise creation process in software engineering (SE) education. This study explores the use of advanced LLMs, including *GPT-4* and *LaMDA*, for automated programming exercise creation in higher education and compares the results with related work using *GPT-3.5-turbo*. Utilizing applications such as ChatGPT, Bing AI Chat, and Google Bard, we identify LLMs capable of initiating different exercise designs. However, manual refinement is crucial for accuracy. Common error patterns across LLMs highlight challenges in complex programming concepts, while specific strengths in various topics showcase model distinctions. This research underscores LLMs' value in exercise generation, emphasizing the critical role of human supervision in refining these processes. Our concise insights cater to educators, practitioners, and other researchers seeking to enhance SE education through LLM applications.

**Index Terms**—AI-Generated Exercises, Large Language Models, Programming Exercises, Software Engineering Education

## I. INTRODUCTION

The field of software engineering (SE) education continually seeks innovative approaches to enhance the learning experience and engage students in meaningful ways. Also, lecturers search for ways to improve their teaching and processes. Creating exercise sheets and assignments is often time-consuming for lecturers, and creating new exercises regularly can be cognitively exhausting. To reduce the effort of creating new exercises, the automatic question generation (AQG) concept was explored [1]. However, AQG research in the field of education is still narrow [2]. We started using large language models (LLMs) to generate self-assessment quizzes in software engineering education [3]. Nevertheless, there seems great potential for also creating programming exercises using AQG. In previous work, we introduced automated-generated programming exercises into the educational context using ChatGPT with *GPT-3.5-turbo* [4]. Recently, the emergence of AI has ushered in new possibilities for automating educational content creation [5], [6].

In this study, we explore the potential of LLM applications such as ChatGPT, Bing AI Chat, and Google Bard based on *GPT-4*, *LaMDA* in the context of programming exercise generation for higher education and compare against our previous *GPT-3.5-turbo* results [4] as baseline. Traditionally, creating

programming exercises that effectively challenge and stimulate students' problem-solving skills has demanded considerable time and effort from lecturers. Integrating LLMs into this process introduces an intriguing opportunity to streamline exercise creation while maintaining the pedagogical integrity and envision a future where a significant portion of programming exercises can be generated with reduced manual intervention. This leads to our research questions for this study:

**RQ1:** “How do ChatGPT with GPT-4, Bing AI Chat, and Google Bard perform in generating programming exercises compared to ChatGPT with GPT-3.5-turbo concerning the quality of the exercises and process simplicity?”

**RQ2:** “Which of the LLM applications is fitted best for lecturers to generate programming exercises?”

The objective is to investigate the feasibility and effectiveness of using these LLM applications to create programming exercises. We intend to offer educators (e.g., lecturers), practitioners, and researchers a detailed perspective on the role of LLM applications in programming education. By analyzing the quality, diversity, and accuracy of the exercises produced, we aim to uncover the strengths and limitations of each LLM in this specific context and compare them to each other. Our investigation delves beyond the theoretical potential of LLMs and into the practicalities of their implementation. We scrutinize the exercise generation process to identify common patterns of error and challenges associated with automated generation. Furthermore, we explore the extent of manual intervention required to refine LLM-generated exercises, thus providing a comprehensive understanding of the dynamics between automated assistance and human oversight. By presenting empirical insights into the capabilities of these models, we hope to contribute valuable insights that inform decisions regarding integrating LLMs into programming curricula.

## II. GENERATION OF EXERCISES

### A. ChatGPT with GPT-3.5-turbo as Baseline

In previous work, we investigated generating exercises for a programming course using ChatGPT with *GPT-3.5-turbo* [4]. We considered a beginner to intermediate-level programming

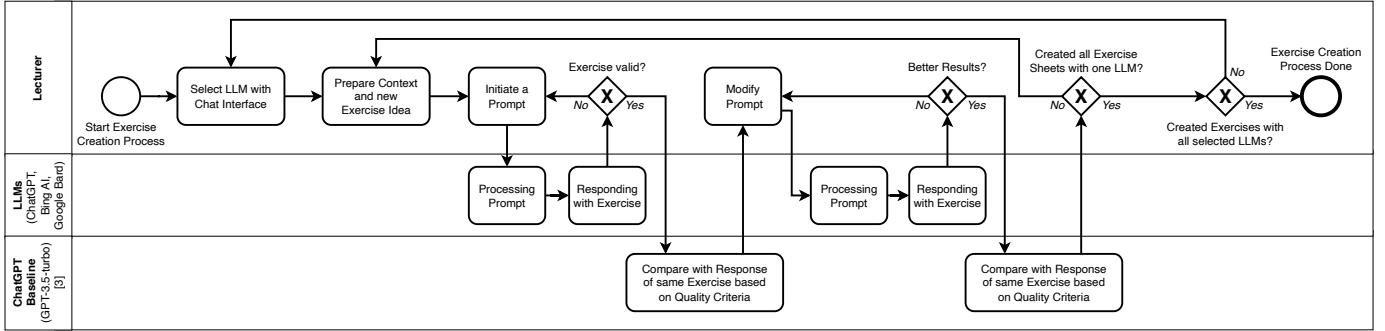


Fig. 1. BPMN of our process to investigate the use of LLM applications for programming exercise creation.

course focusing on the objects-first approach [7], [8], hence we use the same course scenario in this study.

We created 12 exercise sheets<sup>1</sup> with multiple exercises, each for different topics in such a course. Each exercise sheet required minor manual edits to be applicable in an actual course. We evaluated the exercise sheets concerning their quality for student beginners and whether the students saw that the sheets were mostly AI-generated. The results showed that the generated sheets were helpful to the students and that they did not know the sheets were AI-generated [4].

The quality of generated exercises differs depending on the topic of the exercise. While exercises about control flow structures went extremely well, creating exercises that practiced identifying errors and classifying them correctly was very challenging [4]. In this study, we use the previous results with *GPT-3.5-turbo* as a baseline and examine other LLM applications to see if they achieve better results.

Therefore, we selected four different exercise topics for this study and selected the respective exercise sheets and prompts as a baseline against which to compare. The exercise topics we decided on are (i) exercises that provide the students a description of an API and tasks that the students should solve by using the API, (ii) exercises that practice different control flow structures, (iii) exercises with inheritance, generics and the strategy design pattern, and (iv) exercises that practice the understanding of different types of errors. We chose these topics as (i) is complex regarding a good design, (ii) went very well in the baseline, (iii) is very complex, and the baseline exercises had some semantic incorrectness, and (iv) went very bad in the baseline. In another paper, we investigated further exercise generation and comprehension of UML modeling exercises using *GPT-4* and *DALL·E* [9].

### B. General Process

We followed a similar process compared to the baseline [4]. The process is depicted with a BPMN process in Figure 1. We started by selecting the LLMs we intended to test. The most important criterion was that the LLM had a chat interface through which lecturers could interact. We began each generation by providing the context and exercise topic in our

initial prompt. Based on the first prompt, the LLM creates an initial exercise draft, which we review. We either reject the draft if the exercise is not valid or compare it with the exercise in the baseline. We then modify and specify the prompt with more context, a different scenario, or guide the LLM to what we expected. The LLM then generates an updated draft. If the design does not meet our expectations, i.e., it is a bad design, we start again with a different or better-specified context. If we achieve a better result, we then compare it again with the exercise in the baseline. We repeat the process until all exercise sheets are created with all selected LLMs.

The prompt engineering process was carried out step by step in accordance with Velásquez-Henao et al. [10], starting by defining the goal. The prompt was then designed using best-practice prompt patterns [11], the result was assessed using evaluation criteria [12] and then compared with each other. We used the prompt patterns [11] to design prompts such as:

*“Act as a software engineering professor. I want you to create exercises for an advanced beginner programming course in Java about topics I will write in future messages. The exercises themselves should be formatted with markdown. The source code should be in Java. Multiple files are allowed. Please pay attention to good code style, i.e., follow clean code rules. Please create an exercise about inheritance with the strategy design pattern.”*

All prompts and LLM conversations are stored on Zenodo<sup>2</sup>. To determine the response quality, we evaluated the responses according to Lo’s continuous evaluation criteria [12] for (1) accuracy, (2) relevance, and (3) completeness.

### C. LLM Selection

As we have already investigated the general applicability of LLMs to generate programming exercise sheets [4], we are interested in how the different applications that use the same or different LLMs in the background perform this task. We only consider such LLM client applications that lecturers can use without programming any additional code, e.g., to conduct API calls. Therefore, we will not investigate the use of OpenAI’s GPT API, Azure OpenAI service, Meta’s LLAMA, etc., but focus on ChatGPT, Bing AI Chat, and Google Bard. As the baseline used ChatGPT with *GPT-3.5-turbo*, we focus on ChatGPT with *GPT-4* in this work. According to its

<sup>1</sup><https://github.com/spethso/Programmierprojekt/wiki>

<sup>2</sup><https://doi.org/10.5281/zenodo.8298489>

TABLE I  
COMPARISON OF THE LLM APPLICATIONS AND THE BASELINE BASED ON QUALITY CRITERIA.

Exercise topic	Baseline ( <i>GPT-3.5-turbo</i> )		ChatGPT ( <i>GPT-4</i> )		Bing AI Chat ( <i>GPT-4</i> )		Google Bard ( <i>LaMDA</i> )	
	# of Prompts	Quality	# of Prompts	Quality	# of Prompts	Quality	# of Prompts	Quality
Control flow structures	few (1 – 4)	good	few (1 – 3)	good	few (1 – 4)	decent	few (1 – 3)	decent
API understanding and usage	decent (9)	good	few (4)	good	N.A.	N.A.	N.A.	N.A.
OOP - Inheritance, Generics, etc.	few (2 – 4)	decent	few (1 – 3)	good	few (1 – 3)	bad	few (2 – 3)	bad
Error finding and classification	many (?)	bad	few (4)	good	N.A.	N.A.	N.A.	N.A.

documentation, Bing AI Chat uses *GPT-4* as well. However, Google Bard uses Google’s *LaMDA* model. As we could not generate exercises with the web version of Bing AI Chat, we used the version available in Skype.

#### D. Exercise Generation using Different LLMs

As mentioned, we use ChatGPT with *GPT-4*, Bing AI Chat with *GPT-4*, and Google Bard with *LaMDA* to generate exercises for the four topics described in subsection II-A. In the following, we describe our experiences and specific observations when using these applications. We describe the quality of the exercises and usability of the applications without describing every detail. Table I depicts a summary comparison of the three applications and the baseline for the four chosen topics. The table provides information on the number of prompts required to obtain a reasonably acceptable response from the LLMs. We considered up to five prompts to be “few”, up to ten to be “decent”, and everything above that to be “many”. We could not determine the exact number of prompts for the baseline error-finding exercise. However, *GPT-3.5-turbo* required a large number of prompts and achieved poor results. We used the criteria described in subsection II-B to determine the quality of the generated exercises and summarized them as “bad”, “decent” and “good”. In the following paragraphs, the various exercise topics are analyzed in detail, and their quality is described:

1) *Control flow structures*: This exercise sheet aims to practice control flow structures. As the course focused on the objects-first approach, the exercises should follow an OO style, i.e., using classes and objects in the main method [4].

In the baseline, exercises for different control-flow structures were created with only a few prompts. While the exercises are of good quality, they often lack an object-oriented focus. Nevertheless, some exercises for if-else and switch-case fulfill this requirement and provide an acceptable OO design.

While ChatGPT with *GPT-4* created some standard control-flow exercises, e.g., Fibonacci numbers or password checking, it could also generate less typical tasks. However, most tasks lack an acceptable OO design and often use Java I/O to retrieve user input. Nevertheless, we could create most exercises within one to four prompts. Notably, ChatGPT with *GPT-4* often provided the control flow structures already in a start code, which had to be excluded by us via an additional prompt. We did not see this behavior in the baseline.

We were able to generate basic control-flow structure exercises with Bing AI Chat. However, the exercises did not

follow any OO design and did not strictly vary from standard examples. Furthermore, Bing AI Chat used Java I/O even after explicitly stating not to use Java I/O.

Also, Google Bard primarily generated standard examples, e.g., grades and weekdays. We could not create control-flow structure exercises that follow an OO design.

In summary, all three applications can generate decent control-flow structure exercises without OO design. However, we experienced that ChatGPT outperforms Google Bard and Bing AI Chat in terms of creativity, OO design, scenarios, and exercise quality. Notably, we noticed that the baseline performed better than Google Bard and Bing AI Chat.

2) *API understanding and usage*: This exercise sheet aims to practice the understanding and usage of Java APIs. The exercise should provide a textual description of an API, i.e., not providing source code. The students should understand this API and use it to implement different scenarios or methods [4].

In the baseline, generating basic and acceptable exercises for this topic was possible with only a few prompts. However, more detailed change requests or extensions often required many prompts and would have been more easily injected into the exercise manually. The exercises were creative in design and API but often followed a similar structure.

Also, ChatGPT with *GPT-4* could create good-quality API exercises. In comparison to the baseline, we required fewer prompts for acceptable exercises. Nevertheless, ChatGPT with *GPT-4* created similar scenarios and APIs as the baseline.

Using Bing AI Chat, we could not create exercises for this topic that satisfy our requirements. The generated exercises provided the source code directly instead of an API description, even though we explicitly stated otherwise. Furthermore, instead of calling Java class APIs, some tasks expected the calling of remote APIs, which is not intended for a beginner to intermediate programming course.

Like Bing AI Chat, Google Bard could not create exercises for this topic. Google Bard did not have a class API to consume and use, but it had tasks to program an entire class based on a given description or tasks to call a remote API.

In summary, only ChatGPT with *GPT-3.5-turbo*, i.e., the baseline, and with *GPT-4*, could generate exercises that fulfill this topic’s requirements. Using *GPT-4*, we experienced higher creativity and required fewer prompts.

3) *Inheritance, polymorphism, and generics*: This exercise sheet aims to practice more complex OO concepts such as inheritance, polymorphism, dynamic binding, and generics [4].

We also try to create an exercise for the strategy design pattern as an example for the composition over inheritance concept.

In the baseline, exercises for the complex OO concepts mentioned above were created successfully.

Depending on the concept, the quality varied. Some exercises looked correct on the first view but showed semantic incorrectness when trying to solve them. Using one prompt, the baseline could also create a strategy design pattern exercise. In general, the baseline required only a few prompts.

ChatGPT with *GPT-4* generated well-designed exercises for this topic's concepts. However, the exercises use standard examples similar to the baseline. In contrast to the baseline, ChatGPT with *GPT-4* did create Java Generics exercises that were not only a wrapper above Java Collections. Also, the strategy design pattern exercise was created using one prompt. We could not find semantic errors as in the baseline.

Bing AI Chat could also create decent exercises with standard examples for this topic. However, the tasks for Java Generics use either a generic pair class or Java Collections and are, thus, not creative. Additionally, the strategy design pattern exercise did not have a clear and straightforward structure.

Google Bard was able to create exercises for this topic. These exercises follow standard examples, and the Generics exercise uses Java Collections instead of interesting new scenarios. Furthermore, the Generics exercise was semantically broken as the methods contained infinite recursions and an attribute of the child class in a parent class. Google Bard could create a strategy design pattern exercise, but the exercise was generic and not creative with a concrete scenario.

In summary, we could create decent to acceptable exercises for this topic with all three applications similar to the baseline. However, all exercises were primarily created based on standard examples. Nevertheless, we experienced that ChatGPT with *GPT-4* offered the highest quality for the exercises.

4) *Error finding and classification:* This exercise sheet aims to practice identifying errors in existing source code and classifying them, i.e., mapping them to lexical, syntactical, static semantical, and dynamic semantical error classes [4].

In the baseline, there were difficulties creating correct and well-defined exercises on this topic. ChatGPT with *GPT-3.5-turbo* often created correct or nearly correct source code. In case of errors, the baseline often misclassified them. Especially for lexical errors, the baseline seemed not to know the concept.

While ChatGPT with *GPT-4*, in general, could create good exercises for this topic, it still lacks an understanding of lexical errors and misclassified errors of other types as lexical. However, after a short explanation, ChatGPT could apply a correct understanding to the generated exercises. Therefore, we could generate exercises for this topic in only a few prompts.

Bing AI Chat was unable to generate this specific type of exercise. It became apparent that the model was limited in constructing programming exercises that required error identification and correction. This finding underscores the complexity of accurately modeling syntactic and logical errors within code and highlights the challenges inherent in training LLMs to capture such intricacies.

While Google Bard generated exercises with correct source codes, unfortunately, it did not inject intentional errors into the code. This made the generated exercises mostly unusable.

In summary, only ChatGPT with *GPT-4* could create correct exercises with acceptable quality and amount of prompts.

5) *Usability for lecturers:* While the results were decent for most exercise topics, the applications' usability vastly differs.

*Chat history and prompt editing:* In ChatGPT, each prompt can be edited, creating a new branch in the chat history. Therefore, the resulting chats are not linear but follow a tree structure. This allows the lecturer to come back to a prompt and refine it to update the exercise's draft differently, which can be very helpful if some prompts lead to worse results than expected. Instead of repairing the draft within the same branch, the lecturer returns to an earlier prompt and continues creating the exercise. In contrast, Bing AI Chat and Google Bard offer only a linear chat history, and Google Bard does not allow any editing of prompts except for the last prompt, making changes in an earlier state difficult or impossible.

*Structure:* ChatGPT and Google Bard allow the lecturer to create a separate chat for each exercise sheet or single exercise. The lecturer can rename these chats to identify them more easily later. However, Bing AI Chat only offers one chat to work with, creating multiple exercises in the same history.

*Sharing:* ChatGPT and Google Bard offer links to share a chat history with others, e.g., exercises with other lecturers.

*Output format:* As in the baseline, we aimed to create our exercises in markdown format. While ChatGPT and Google Bard can generate exercises in markdown and render them within the chat, Bing AI Chat could not provide markdown. However, we have not tried other formats such as *LaTeX*.

Overall, in our opinion, ChatGPT offers the best usability to lecturers as it allows the most fine-grained editing while keeping the entire chat history as a tree-like structure, allowing them to switch between the branches. Using a separate chat for each exercise sheet and formatting is possible and allows renaming the chats. Furthermore, the exercises were formatted in markdown as requested and rendered. Google Bard and Bing AI Chat lack some criteria and do not offer additional features to improve their usability.

### III. LESSONS LEARNED AND THREATS TO VALIDITY

#### A. Lessons Learned

There are some lessons learned for generating programming exercises with LLMs: (1) ChatGPT outperforms the other LLM applications regarding results and usability. (2) Even though ChatGPT and Bing AI Chat both use *GPT-4*, the quality of the results differs a lot. (3) Google Bard offers confident explanations that often are incorrect, e.g., for error identification and classification. (4) The LLM applications often provide a starter code in the control flow structures exercise, already containing the control flow structure to practice. (5) Context-switches and editing of prompts work best in ChatGPT due to the branching of lecturer input and providing multiple chats. (6) Exercise scenarios are very similar in all applications.

## B. Threats to Validity

We identified three main threats to validity. (1) In our study, we were investigating the use of the three applications only on four programming exercise topics, which we selected based on the experiences in the baseline study. (2) For each topic, we created around five exercises. Therefore, a larger study might be required to make stronger empirical-based assumptions about the quality of the generated exercises. (3) We did not evaluate the exercises' quality in an actual course but only by manual review. However, despite the stated threats, we believe that the lessons learned and insights presented are valuable and useful for educators, practitioners, and other researchers.

## IV. RELATED WORK

The research by Sarsa et al. [13] focused on exploring the capabilities of generating programming exercises with large language models, focusing on OpenAI Codex. Similar to this paper, the goal was to use these capabilities to generate programming exercises with associated elements such as sample solutions, test cases, and code explanations [13]. However, the authors only obtained results with OpenAI's outdated Codex model, where this paper analyzes the differences between the various large language models and presents recommendations for specific programming exercise types.

Nguyen et al. [14] used an approach to generate questions with LLMs. They aimed to generate and assess questions using *GPT-3*. Even though *GPT-3* is an outdated model, they also focused on higher education in the field of data science. They assessed the created questions using automatic scoring by Google *GPT-3* and manual review [14]. However, since we focus on the generation of exercises and differences in using state-of-the-art LLMs, the objectives of the studies differ.

The generation of programming exercises was also investigated by Lu et al. [15]. Their approach was to develop a question-generation system that combined semantic and syntactic elements. The main objective was to compare the quality of questions generated by the machine with those generated by experts [15]. However, compared to these related works, we focus on LLM applications accessible to lecturers and practitioners without having to build own custom generators. Moreover, we focus on more complex programming exercises in higher education, which the other works mostly did not do.

## V. CONCLUSION

In this paper, we investigated the use of ChatGPT with *GPT-4*, Bing AI Chat, and Google Bard to generate exercises for a beginner to intermediate-level programming course scenario. We compared the results against each other and used ChatGPT with *GPT-3.5-turbo* as a baseline from related work regarding the quality of the exercises and usability. Our study focuses on four different exercise topics that went very well or poorly with *GPT-3.5-turbo*, were very challenging in design, or were complex concepts. Our results showed that ChatGPT with *GPT-4* fitted best, followed by ChatGPT with *GPT-3.5-turbo*. Google Bard provides good usability for lecturers but had worse results than ChatGPT with *GPT-3.5-turbo*. Bing AI Chat

was cumbersome to use and did not provide particularly good results for the used input prompts. Educators, practitioners, and other researchers can use the results of this study to create own exercises using the LLMs mentioned, taking into account the strengths and weaknesses described here, and thus achieve the desired result more efficiently. In general, ChatGPT and Google Bard could easily generate good programming exercises. They especially help in creating exercise variants and good first drafts. However, in future work, LLMs without a graphical user interface, which may be specialized in creating exercises, could also be investigated. Nevertheless, generated exercises still usually require manual edits and cross-checking.

## ACKNOWLEDGMENT

We acknowledge the support of the Ministerium für Wissenschaft, Forschung und Kunst Baden-Württemberg (MWK, Ministry of Science, Research and the Arts Baden-Württemberg under Az. 33-7533-9-19/54/5) in Künstliche Intelligenz & Gesellschaft: Reflecting Intelligent Systems for Diversity, Demography and Democracy (IRIS3D) and the support by the Interchange Forum for Reflecting on Intelligent Systems (IRIS) at the University of Stuttgart.

## REFERENCES

- [1] G. Kurdi, J. Leo *et al.*, "A Systematic Review of Automatic Question Generation for Educational Purposes," *International Journal of Artificial Intelligence in Education*, vol. 30, no. 1, pp. 121–204, Mar 2020.
- [2] T. Steuer, L. Bongard *et al.*, "On the Linguistic and Pedagogical Quality of Automatic Question Generation via Neural Machine Translation," in *Technology-Enhanced Learning for a Free, Safe, and Sustainable World*. Cham: Springer International Publishing, 2021, pp. 289–294.
- [3] N. Meißner, S. Speth, J. Kieslinger, and S. Becker, "EvalQuiz – LLM-based Automated Generation of Self-Assessment Quizzes in Software Engineering Education," in *Software Engineering im Unterricht der Hochschulen 2024*. Bonn: GI e.V., 2024, pp. 53–64.
- [4] S. Speth, N. Meißner, and S. Becker, "Investigating the Use of AI-Generated Exercises for Beginner and Intermediate Programming Courses: A ChatGPT Case Study," in *2023 IEEE 35th International Conference on Software Engineering Education and Training (CSEE&T)*.
- [5] B. du Boulay, "Artificial Intelligence as an Effective Classroom Assistant," *IEEE Intelligent Systems*, vol. 31, no. 6, pp. 76–81, 2016.
- [6] G.-J. Hwang, H. Xie *et al.*, "Vision, challenges, roles and research issues of Artificial Intelligence in Education," *Computers and Education: Artificial Intelligence*, vol. 1, p. 100001, 2020.
- [7] S. Speth, "Teaching Object-Oriented Programming with the Objects-first Approach: An Experience Report," in *Proceedings der SEUH 2023*. Gesellschaft für Informatik, Bonn, 02 2023, pp. 35–42.
- [8] B. Meyer, *Touch of Class*. Springer, 2009, vol. 51.
- [9] S. Speth, N. Meißner, and S. Becker, "ChatGPT's Aptitude in Utilizing UML Diagrams for Software Engineering Exercise Generation," in *2024 IEEE 36th International Conference on Software Engineering Education and Training (CSEE&T)*.
- [10] J. D. Velásquez-Henao, C. J. Franco-Cardona, and L. Cadavid-Higuera, "Prompt Engineering: a methodology for optimizing interactions with AI-Language Models in the field of engineering," *DYNA*, vol. 90, no. 230, p. 9–17, Nov. 2023.
- [11] J. White *et al.*, "A Prompt Pattern Catalog to Enhance Prompt Engineering with ChatGPT," 2023.
- [12] L. S. Lo, "The CLEAR path: A framework for enhancing information literacy through prompt engineering," *The Journal of Academic Librarianship*, vol. 49, no. 4, p. 102720, 2023.
- [13] S. Sarsa, P. Denny *et al.*, "Automatic generation of programming exercises and code explanations using large language models," in *Proceedings of the 2022 ACM Conference on International Computing Education Research - Volume 1*, ser. ICER '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 27–43.
- [14] H. A. Nguyen, S. Bhat *et al.*, "Towards Generalized Methods for Automatic Question Generation in Educational Domains," in *Educating for a New Future: Making Sense of Technology-Enhanced Learning Adoption*. Cham: Springer International Publishing, 2022.
- [15] O. H. T. Lu, A. Y. Q. Huang *et al.*, "Expert-authored and machine-generated short-answer questions for assessing students learning performance," *Educational Technology & Society*, vol. 24, no. 3, 2021.



# Systematic Teaching of UML and Behavioral Diagrams

Anja Metzner  
Technical University of Applied Sciences Augsburg  
Faculty of Computer Science  
Augsburg, Germany  
anja.metzner@tha.de

**Abstract**— When studying software engineering, learning to create UML diagrams is crucial. Similar to how an architect would never build a house without a building plan, designing software architectures is important for developing high-quality software. UML diagrams are a standardized notation for the visualization of software architectures and software behavior.

The research question that inspired this work was how to effectively evaluate hand-drawn diagrams without relying on model parsers. The findings of this investigation are presented in this paper. This article discusses the systematic acquisition of skills required for creating UML diagrams. Especially well-formed activity diagrams are one highlight.

Additionally, the paper provides a variety of exercises. The exercises use recommended question types. The more unusual question types are related to images, such as questions about image annotation, finding hotspots on an image and positioning a target on an image. All the demonstrated exercises are suitable for both digital and handwritten training or exams.

**Keywords**— UML modeling, systematic, competence-based teaching, well-formed activity diagrams

## I. INTRODUCTION

Are you teaching modeling with UML [1]? Are you responsible for evaluating UML diagrams created by students? How can we professionals evaluate drafted activity diagrams fairly and grade appropriately? How can we effectively train and assess student comprehension of sequence diagrams? Software engineering teachers are confronted with questions similar to those above. Their goal is to design appropriate trainings and exams.

This paper proposes strategies for improving the expertise of students in creating UML diagrams through the use of a step-by-step approach. I propose a systematic and reliable process for proofreading and assessing students' diagrams without using parsers (e.g. [17, 18, 8, 9]). Suitable examples of exercises are provided for demonstrating the strategies.

### A. Methodology

Teaching UML modeling is challenging. One challenge is the broad applicability of UML beyond software engineering. Thus, this work concentrates on software modeling. In the context of Germany's educational landscape, I outline a competency-based approach for teaching UML. Additionally, I address controversial aspects of standard UML notation and propose solutions. The rest of the paper introduces an approach to teaching UML through relevant exercises.

### 1. Related Work

This paper primarily focuses on **modeling exercises**. While research on activating teaching approaches in software engineering exists [20], other studies discuss best practices for pedagogy and subject-specific teaching of modeling [21]. These practices address issues such as '*tailoring the development process*', '*definition of sequencing UML artefacts*', '*constant feedback from/to the participants*' and '*conducting experiments*'. Authors advocating best practices in software engineering recommend incorporating projects or

case studies into courses [22]. The techniques presented in this paper align with these suggested best practices.

In order to focus on exercises, it is essential to consider research on **common mistakes made by students** when working with UML. This paper takes into account up-to-date investigations into frequently occurring errors (refer to [19]). While decision nodes in activity diagrams consistently pose challenges for learners—ranging from '*missing decision node description*' to '*multiple control flows enter one action node*'—loops are not explicitly addressed in the mentioned study. Therefore, alongside the competency-based teaching approach, this paper proposes solutions for modeling and teaching branches and loops in behavioral diagrams.

### 2. The Approach of Dealing with Competency-Based Education

In the context of the German university landscape, there is a prevailing preference for competency-based teaching, as elaborated in the expert report on competency orientation in study and teaching by the German Rectors' Conference (Hochschulrektorenkonferenz - HRK) in 2012 [2]. As a result, numerous degree programs and module handbooks have been designed with a competency-based approach. [3, 4]. Based on Bloom's concepts (1956) [5] a taxonomy matrix has been described at the HRK conference. It was adapted by Anderson and Krathwohl in 2001 [6], and was later expanded once again by HRK in 2013 [3]. This approach follows the "*constructive alignment*" principle (invented by Biggs and Chow [7]).

However, categorizing the approaches presented in this work within the "HRK taxonomy matrix" would depend on the specific objectives of each teaching session. Consequently, only the taxonomy levels' labeling based on Bloom's principles are used, ranging from **remember to create**. This simplifies easy integration into current study programs.

### 3. Dealing with Criticism of the Standard Notation

UML has emerged as the standard for modeling. However, it is essential to recognize that UML does not offer a universal solution to all modeling challenges. Valid criticisms exist regarding this notation. For instance, the modeling of processes in activity diagrams relies on outdated flowcharts (i.e. program flowcharts), often resulting in complex diagrams. This complexity increases when processes contain many nested and repetitive elements. Given UML's status in the modeling domain, I agree that students must be thoroughly prepared to deal effectively with those aspects. Consequently, Chapter III.B discusses some important issues and proposes solutions to enhance proper training for students.

### B. Content of this Work

Chapter II of this paper outlines the essential skills for UML modeling through the six taxonomy levels specified in [3]. The approaches recommended in this paper are shown in Chapter III. Concrete examples are provided, that are relevant to the practice of teaching. In Chapter VI an evaluation from lectures following the suggested techniques is described. Chapter V concludes with a summary and future work.

TABLE I. COMPETENCY CLASSIFICATION IN UML MODELING

No.	Competencies	Level
A	Identifying and naming of UML elements	1-Remember
B	Recognizing diagram types	
C	Reading diagrams	2-Understand
D	Inserting, modifying or deleting elements (e.g., multiplicities, stereotypes, visibilities)	3-Apply
E	Properly applying relationship elements	
F	Drawing control structure elements (such as sequence, branching, and loop)	
G	Creating comprehensive diagrams	
H	Identifying incorrect or allowed elements	4-Analyze
I	Checking UML syntax for purpose (correct arrow type, correct symbol)	
J	Checking order of elements	
K	Verifying control structure elements (i.e., sequence, branching, and loop)	
L	Analyzing diagrams (e.g. sensibility, meaningfulness, and correctness)	
M	Assessing the quality of UML diagrams (e.g. programmable)	
N	Evaluating UML diagrams according to problem definition	5-Evaluate
O	Formulating and achieving modeling goals (To achieve various modeling objectives, one should be familiar with the application and purpose of UML diagram types, trained in their modeling, and capable of utilizing best practices).	6-Create

## II. REQUIRED COMPETENCIES FOR MODELING WITH UML

The proposed approach to teach modeling follow the described six taxonomical levels: **1-remember, 2-understand, 3-apply, 4-analyze, 5-evaluate, and 6-create**.

A proper classification of competencies for designing UML diagrams into the suggested taxonomy levels is not a straightforward transaction, so it is advisable for teachers to carefully reconsider such categorization themselves. This paper uses for demonstration the distribution of competencies presented in Table I.

TABLE II. GENERAL AND DIAGRAM-SPECIFIC COMPETENCIES

General Aspects	
(A)	Identifying UML-elements
(B)	Recognizing diagram types
(C)	Reading diagrams
(D)	Inserting, modifying, and deleting elements (e.g., multiplicities, stereotypes, visibilities)
(E)	Properly applying relationship elements
(F)	Drawing control structure elements
(G)	Creating a comprehensive diagram
(H)	Identifying incorrect/allowed elements
(I)	Checking UML syntax for purpose (correct arrow type, correct symbol)
Specifics of Individual Dynamic Diagram Types:	
<i>Activity diagram:</i>	(J) Checking element order
	(F) Draw and verify (K) control structure elements (i.e., sequence, branching, and loops)
	(L, M) Well-formedness (e.g. merging, or multiple end symbols)
	(L, M) Clear labeling
<i>Sequence diagram</i>	(J) Verify element order
	(F, K, L, M) Train consistency between activity and sequence diagrams (advantage: ability of distinguishing between each other)
	(F) Practice nesting control structure elements

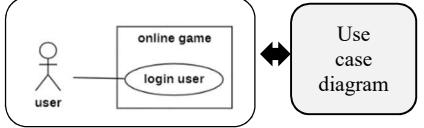
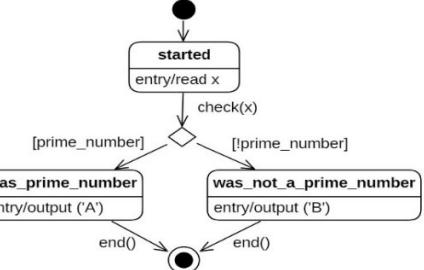
In this work, Chapter II demonstrates training aspects, showing how to concretely achieve general (i.e. for all diagram types the same) and diagram-specific competencies listed in Table II. Each list item is aligned with the **uppercase letters** in Table I. Thus, I discuss the competencies outlined in the both tables.

## III. PROPOSED APPROACHES WITH EXAMPLES

### A. General Aspects

The techniques outlined in Table II can be applied to all UML diagram types. A teaching example is provided for each required competence (Table III). Point grading scales (abbr. PGS) can be used, which can be easily applied.

TABLE III. QUESTIONING GENERAL ASPECTS IN UML DIAGRAMS

Competence	Method Proposal	Level of Taxonomy	Evaluation Option
<i>Identifying UML-elements (A)</i>	<b>Knowledge questions</b>	<b>Remember</b>	<b>PGS</b>
	Concrete example: 'A filled diamond is used to represent a composition in a class diagram. (True / False)', Answer: True.		
<i>Recognizing diagram types(B)</i>	<b>Matching questions</b>	<b>Remember</b>	<b>PGS</b>
	To teach learners about the distinctions between diagram types, instructors may use a UML Memory game, either in print, or by utilizing H5P task library [11], or other tools. Concrete example: 'Always look for two matching cards (see Figure 1). '		
			
	Fig. 1. UML-Memory Game		
<i>Reading diagrams (C)</i>	<b>Questions about image annotations</b>	<b>Understand</b>	<b>PGS</b>
	A useful strategy for enhancing the learners diagram-reading skills is to work with input and output values. This approach offers flexible exercises by using different values and is illustrated through an exercise with a state machine. Concrete example: 'Let us suppose that you enter the number 5 into the 'Number Code Machine' state machine diagram depicted in Figure 2. In this scenario, what letter would the machine produce as its output?', Answer: A.		
			
	Fig. 2. State machine diagram with input and output		
<i>Inserting, modifying, and deleting elements (D)</i>	<b>Questions about image annotations</b>	<b>Apply</b>	<b>PGS</b>
	Concrete example: 'What should be the multiplicity added to Figure 3 for circular marking 1, and what should be the multiplicity added to marking 2 for an accurate representation?', Answer: 1- Value '1' and 2- Value '*'.		
			
	Fig. 3. Question regarding an annotation in an image		
<i>Properly applying relationship elements (E)</i>	<b>Questions about image annotations</b>	<b>Apply</b>	<b>PGS</b>
	Using a use case diagram, it is demonstrated how relationship elements and arrow directions can be queried. Concrete example: 'Given is the incomplete UML use case diagram of Figure 4, which is described in the following. Scenario: A teacher administers an exam, which can either be oral or written. For the oral exam, only one student may participate, while any number of students can participate in the written exam, but there must be at least one participant. Becoming ill during the exam, students report their illness. To complete the exam, the teacher must record the grades.'		

Competence	Method Proposal	Level of Taxonomy	Evaluation Option
	Fig. 4. Use case diagram 'documentation system'		
	Question: The type of relationship that the orange line represents at point (a) in the diagram is a/an _____ -relationship. The arrowhead of this orange line points to use case number UC-_____!. Answer: 'extend', 'UC-1'.		
	In the digital format, the specific question for the learner can appear as depicted in Figure 5 [10].		
	<div style="border: 1px solid black; padding: 5px;"> <p>The type of relationship that the orange line represents at point (a) in the diagram is a/an _____ -relationship. The arrowhead of this orange line points to use case number UC-_____!.</p> </div>		
	Fig. 5. Form of an digital response		
Drawing control structure elements (F)	<b>Creating a comprehensive diagrams</b>	Apply	PGS
	The practice entails drawing control structure elements, such as sequence, branches, and loops (refer to the discussion in Chapter III.B).		
	Concrete example: 'Draw an activity diagram with a while loop', Answer: Please see Figure 6a. Alternatively, a more structured representation with loop nodes can be found in Figure 6b, according to [12].		
	Fig. 6. Activity diagram with a while loop		
Creating a comprehensive diagram (G)	<b>Creating a comprehensive diagram</b>	Apply	PGS
	For modeling novices, it is advisable to commence with basic, illustrative exercises, whereas professionals may rehearse with intricate, supplied specifications.		
	Concrete example: 'Design a use case diagram in which a user has the ability to sign into an online game.', Answer see Figure 7.		
	Fig. 7. Use case diagram 'online game'		
Identifying incorrect/allowed elements (H)	<b>Finding hotspots in an image</b>	Analyze	PGS
	With image hotspots, it is possible to inquire about incorrect or allowed elements in diagrams.		
	Concrete example: 'In the UML notation, which elements are not permitted in the use case diagram? Please mark/click on the NOT allowed elements in Figure 8',		
	Answer: Both associations between the use cases are not permitted. (Associations from the actor to the two lower use cases would need to be added.)		

Competence	Method Proposal	Level of Taxonomy	Evaluation Option
	Fig. 8. Use case diagram 'game engine'		
Checking UML syntax for purpose (I)	<b>Questions about image annotations</b>	Analyze	PGS
	Concrete example: 'A modeler must depict an asynchronous message in the sequence diagram. Which arrow type should be used (see Figure 9), the type of Message1, or the type of Message2 ?' Answer: Message 2		
	Fig. 9. Sequence diagram with an asynchronous message		

## B. Specifics of Individual Dynamic Diagram Types

Aspiring modelers must proficiently depict process and program flows, ensuring that the resulting model can be translated into executable software code. Consequently, this work specifically investigates two essential diagram types: activity diagrams and sequence diagrams.

### 1. Activity Diagrams (F, J, K, L, M)

For a meaningful evaluation of activity diagrams, some guidelines are introduced.

#### a) Guidelines and Reasons for the Guidelines:

As Jacopini showed, in programming, there are three fundamental control structure elements: sequence, branch, and loop [13]. These elements allow modeling of mainly all processes and control flows. As discussed in [15, 23, 14], this work distinguishes between **well-formed** and **abstract** (i.e., non-well-formed) activity diagrams. Well-formed activity diagrams can be easily implemented in programming languages because the guidelines allow for only one end symbol and one token within a single control flow. Exceptions exist in concurrent programs and constructs like 'break' and 'continue.' Loops typically consist of a loop head (diamond symbol), a loop body (actions), and a loop end (merge symbol). An alternative notation using loop nodes and decision nodes is also permissible [12]. Branches begin with a diamond symbol and end with a merge symbol, recognizable in programming languages by the closing parenthesis in if-statements. **Well-formed activity diagrams are closely tied to programming**, while abstract diagrams serve non-programmable processes.

Well-formed activity diagrams are valuable for assessing learners due to **anticipated** solutions and their ability to provide **unambiguous outcomes**. Additionally, they offer a **fair assessment** that can be assigned to all students through a point system. Proficient individuals can **comprehend abstract variations and convert them to well-formed versions**, allowing learners to **actively choose** between the two variants when creating models.

b) *Labeling and Commenting:*

Clear and standardized **labels** enhance the readability and learnability of activity diagrams, facilitating **easier corrections**. For branches and loops, a comment field is essential to specify the control structure type (e.g., If, while, do-while, for) and a clear condition (e.g., if ( $i > 1$ ) indicating  $i$  is greater than 1). Additionally, numbering branches and loops is mandatory, using the same number in both the opening and closing diamond symbols.

c) *Demonstration:*

For demonstration purposes, this paper provides a straightforward sample of a properly labeled and well-formed activity diagram, presented in Table IV.

TABLE IV. INQUIRING SPECIFICS OF ACTIVITY DIAGRAMS

Competence	Method Proposal	Level of Taxonomy	Evaluation Option
<i>Activity diagram (F, J, K, L, M)</i>	<i>Knowledge questions</i>	<i>Analyze</i>	<i>PGS</i>

Concrete example:  
'Is the activity diagram of Figure 10 well-formed?' ,

Fig. 10. Well-formed activity diagram

Answer: Yes (The well-formed activity diagram presented in Figure 10 features red dashed lines encircling the loop and branching to enhance visibility).

TABLE V. CHECKING WELL-FORMEDNESS IN ACTIVITY DIAGRAMS

Competence	Method Proposal	Level of Taxonomy	Evaluation Option
<i>Activity diagram (F, J, K, L, M)</i>	<i>Positioning a target on an image</i>	<i>Apply / analyze</i>	<i>PGS</i>

Fig. 11. NOT-well-formed activity diagram 'Order' [16]

Concrete example:  
'Please arrange the elements provided in Figure 11'

Competence	Method Proposal	Level of Taxonomy	Evaluation Option
			correctly! (Note: Elements may be selected zero, one, or multiple times and may also overlap. An incorrect spot must be marked only once with a suitable element. Each element has a target at the top left, which must be precisely aligned with the spot to be marked).', Answer: see Figure 12.

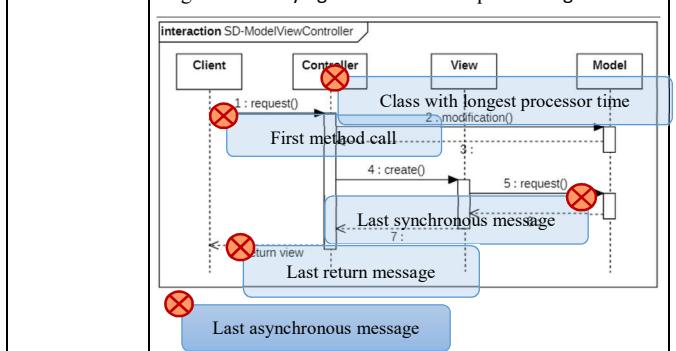
Fig. 12. Solution for the activity diagram 'Order'.

For practicing well-formed diagrams, abstract variations are useful. The objective is to identify non-well-formed elements or elements missing for well-formedness. (see Table V).

TABLE VI. LEARNING SEQUENCE DIAGRAM - TECHNIQUE 2

Competence	Method Proposal	Level of Taxonomy	Evaluation Option
<i>Sequence diagram (F, J, K, L, M)</i>	<i>Positioning a target on an image</i>	<i>Apply</i>	<i>Point grading scale (Evaluation example: One point for each element to be drawn)</i>

Fig. 13. Identifying elements in a sequence diagram



Concrete example - Technique 2:  
The use of this question type is appropriate for assessing diagram comprehension and sequencing accuracy.  
'Please arrange the elements provided below in Figure 13'

Competence	Method Proposal	Level of Taxonomy	Evaluation Option
	correctly! (Note: Elements may be selected zero, one, or multiple times and may also overlap. An incorrect spot must be marked only once with a suitable element. Each element has a target at the top left, which must be precisely aligned with the spot to be marked).', Answer: see Figure 14.		

## 2. Sequence Diagrams (F, J, K, L, M)

This work describes two handsome techniques for teaching the characteristics of sequence diagrams.

a) *Technique 1:* The first approach requires that an activity diagram must exactly match a sequence diagram, meaning that the activity diagram must match the sequence diagram one to one. On one hand, this enhances comprehension of the distinctions between activity diagrams and sequence diagrams. On the other hand, nesting of control flow elements such as branches and loops is easier to learn through sequence diagrams since those elements are nested like boxes. Visual detection of incorrect crossovers is facilitated. Exercises can be found in literatur (e.g. [23, 14]). Teachers should clarify that this is only a lecture approach.

b) *Technique 2:* The second approach to mastering the fundamentals of sequence diagrams involves studying the precise order of elements and the various possibilities that can occur within these diagrams. (see Table VI).

## C. Advanced Competencies

Once the basics have been learned to the level of confident use of diagrams (i.e. taxonomy level: apply), advanced studies can begin with training in analysis based on quality criteria (M), evaluation (N), and advanced creation (O). For high taxonomy levels quality assessments of the work are more meaningful, for which grade scales are appropriate.

The suggestion is to switch at these levels to textual questions, such as '*Based on the given requirements and the provided (or self-created) UML diagram, perform an analysis according to the quality criteria of correctness, meaningfulness, and feasibility.*' or to ask for evaluating a model. Regarding advanced diagram creation one could ask '*Create a software architecture for problem X: Formulate modeling goals and demonstrate how you can achieve them with best practices*'. Finally, finding appropriate boundaries between beginner and expert training might be helpful.

## IV. EVALUATION

Evaluation method: Observation and time measurement. Before and after introduction to well-formedness in my lectures I asked to write an activity diagram with a branch nested in a loop. Once the approach was adopted, measurements revealed students achieve correct solutions up to three times faster (see table VII), with improved quality (e.g. programmable activity diagrams).

TABLE VII. APPROACH EVALUATION

Term	2022/2023	2023	2023/2024
Number of students	90	65	100
Time before introduction (min)	17	15	19
Time after introduction (min)	5,5	6	5
Saved time (%)	32	40	26
Average (%):			33

## V. SUMMARY AND OUTLOOK

This paper outlines a structured approach for students to acquire step-by-step modeling skills. Concrete examples demonstrate how professors can train and assess UML diagrams, whether in digital or written form. The work includes two lists of UML-specific competencies, each

aligned with Bloom's taxonomy levels. Concrete teaching exercises illustrate the value of different approaches.

Future work will explore well-formedness in dynamic process models and compare various modeling methods. Additionally, an open-source web-based toolkit is proposed to serve as a repository for UML modeling exercise patterns, fostering collaboration and evidence-based practices.

## REFERENCES

- [1] "About the UML", OMG, [Online]. Available: <https://www.omg.org/spec/UML>. [Accessed 06 07 2023]
- [2] N. Schaper, O. Reis, J. Wildt, E. Horvath, E. Bender and HRK (Ed.), "Fachgutachten zur Kompetenzorientierung in Studium und Lehre", 2012
- [3] N. Schaper, F. Hilkenmeier and E. Bender, "Umsetzungshilfen für kompetenzorientiertes Prüfen - HRK Zusatzgutachten," Hochschulrektorenkonferenz, 2013
- [4] "Taxonomie-Matrix zur Analyse und Selbstevalution von Hochschullehre (TAMAS)", University Zürich, Bereich Lehre Arbeitsstelle für Hochschuldidaktik (Ed.), [Online]. Available: [http://www.hochschuldidaktik.uzh.ch/instrumente/dossiers/DU\\_Tamas\\_def-1.pdf](http://www.hochschuldidaktik.uzh.ch/instrumente/dossiers/DU_Tamas_def-1.pdf). [Accessed 22 01 2023], 2010
- [5] B. S. Bloom, M. D. Engelhart, E. J. Furst, W. H. Hill, David R. Krathwohl: "Taxonomy of educational objectives: The classification of educational goal. Handbook I: Cognitive domain", David McKay Company, New York, 1956
- [6] L. Anderson and D. Krathwohl, "A taxonomy for learning, teaching and assessing", New York: Longman, 2001
- [7] J. B. Biggs and C. K. Chow, Teaching for quality learning at university: what the student does., Maidenhead: McGraw-Hill, 2011
- [8] M. Brambilla, J. Cabot and M. Wimmer, "Model-driven software engineering in practice", 2.nd Edition., Morgan and Claypool Life Sciences, 2017
- [9] A. Roques, "Plant UML," [Online]. Available: <http://www.plantuml.com>. [Accessed 23 12 2023]
- [10] "Moodle - Online learning delivered your way," Moodle.com, [Online]. Available: <https://moodle.com/>. [Accessed 06 07 2022]
- [11] "H5P," [Online]. Available: <https://h5p.org/>. [Accessed 19 07 2022]
- [12] C. Rupp, S. Queins and Sophisten-Group, "UML 2 glasklar", Carl Hanser, 2012
- [13] C. Böhm and G. Jacopini, "Flow diagrams, Turing machines and languages with only two formation rules," *Communications of the ACM*, Vol. 9, Nr. 5, pp. 366-371, 1966
- [14] A. Metzner, "Software Engineering - kompakt", Carl Hanser, 2020
- [15] A. Holl and K. Grünauer, "Business process modeling", [Online]. Available: <https://www.in.th-nuernberg.de/professors/holl/Personal/HollHome.htm>. [Accessed 21 12 2023], 2008
- [16] "Aktivitätsdiagramme, Figure 14,"University Marburg, [Online]. Available: <https://www.mathematik.uni-marburg.de/~hesse/uml/Aktivitaetsdiagramm.html>. [Accessed 01 08 2023]
- [17] T. Ahmad, J. Iqbal, A. Ashraf, D. Truscan, I. Porres, "Model-based testing using UML activity diagrams: A systematic mapping study", Computer Science Review 33, 2019
- [18] F. Ciccozzi, I. Malavolta, B. Selic, "Execution of UML models: a systematic review of research and practice", Software & Systems Modeling 18, 2019
- [19] S. Chren, B. Buhnova, M. Macak, L. Daubner, B. Rossi, "Mistakes in UML Diagrams: Analysis of Student Projects in a Software Engineering Course", IEEE/ACM 41st Int. Conf. on Software Engineering: SE Education and Training (ICSE-SEET), 2019
- [20] N. Unkelos-Shpigel, J. Sheidin, M. Kupfer, "Climb Your Way to the Model: Teaching UML to Software Engineering Students", Advanced Information Systems Engineering Workshops (CAiSE 2019), LNBP, vol. 349, 2019
- [21] L. Kuzniarz, M. Staron, "Best Practices for Teaching UML based Software Development", 8th MoDELS Satellite Events 2005, LNCS 3844, Springer, 2006
- [22] W. P. P. Filho, "Process Issues in Course Projects", In the Proceedings of 27th Int. Conf. on Software Engineering, St. Louis, Mo, USA, 2005
- [23] P. Forbrig, "Objektorientierte Softwareentwicklung mit UML", Carl Hanser, 2024



# ScrumBoard: A Project Management Tool Purpose-Built for Software Engineering Education

Matthew Minish

*Department of Computer Science  
and Software Engineering  
University of Canterbury  
Christchurch, New Zealand  
matthew.minish@pg.canterbury.ac.nz*

Fabian Gilson

*Department of Computer Science  
and Software Engineering  
University of Canterbury  
Christchurch, New Zealand  
fabian.gilson@canterbury.ac.nz*

Matthias Galster

*Department of Computer Science  
and Software Engineering  
University of Canterbury  
Christchurch, New Zealand  
mgalster@ieee.org*

**Abstract**—This paper recounts our experiences with the ‘ScrumBoard’, a custom-built open-source digital project management tool specifically designed for software engineering project courses. We discuss how it has supported experiential learning pedagogy and assessment over multiple iterations of a two-semester, 3<sup>rd</sup> year project course, having now been used extensively by more than 200 students. While offering a similar experience to commercial software project management tools, we reflect on how the ScrumBoard has supported our ability to assess students more accurately and efficiently, and how the tool has enabled us to identify problematic work practices in real-time, allowing for earlier interventions from teaching staff and an improved learning experience for students.

**Index Terms**—Higher education, project course, project-based learning, Scrum, agile software development

## I. INTRODUCTION

Software engineering education heavily utilizes experiential learning such as capstone projects to prepare graduates for entry into the workforce [1], [2]. Project courses allow students to work in teams to develop complex software applications so they can experience the full software development life-cycle. Amongst the methods used to teach software development in higher education, Scrum is a widely used framework [3]. Unfortunately, commercial project management tools that support Scrum often come with a prohibitively high price-tag for educational contexts, and from our observations, may provide an experience that is overwhelming for students and frustrating for educators, e.g., complex permission system, unclear jargon beyond core Scrum vocabulary, insufficient data regarding students’ learning.

In this paper we describe our experiences using the ‘ScrumBoard’, a purpose-built project management tool used by more than 200 students over two and a half years in software engineering project courses at the University of Canterbury. The tool offers all the functionality required for both students to manage their work and educators to run all aspects of a project course, such as assessing students and implementing educational practices like self-reflections or peer-assessments. The tool is focused on supporting pedagogy and student learning, while still being similar enough to industry standard tools so that students can directly transfer their experience using the ScrumBoard into their graduate roles.

The ScrumBoard is an open source tool available at: <https://gitlab.com/uc-scrumboard/scrumboard>.

## II. BACKGROUND AND MOTIVATION

Experiential learning is a cornerstone of lifelong learning outcomes for students [4]. Furthermore, it is crucial to train future software engineers to be industry-ready, including the ability to work consistently, and track the progression of their work [2]. To this end, students are expected to get familiar with project management tools and vocabulary [5]. However, on top of typical agile project management features and terminology, there is a need for an individualised approach to monitor students’ performance and engagement in the project work as every student’s learning journey is different.

Various purpose-built project management tools have been developed in educational context, e.g., [6]–[8]. However, most of these tools have a look and feel that is too different from commonly used (commercial) tools in industry, hindering the ability of students to transfer their knowledge. On the other hand, commercial or open-source tools such as Jira or Agilefant miss crucial features to enable educators to fully evaluate students’ learning, including their ability to apply the Scrum framework and its values. Notably, the main issues when using existing tools include: the inability to gather data on individual student learning; confusing terminology; inability to record the outcome of all Scrum events, particularly Daily Scrums and Sprint Reviews; and, the lack of flexibility to add educational features such as the ability for students to write self-reflections or peer-evaluations, or identifying engagement issues.

## III. SCRUMBOARD

The ScrumBoard was built for the Scrum framework. Other project management tools are typically framework agnostic and provide an abstracted taxonomy useful for generalizability to multiple organisations. However this can be confusing to students learning agile principles and concepts for the first time. Therefore, the ScrumBoard constrains all artefacts to Scrum terminology, with few additions such as user stories, acceptance criteria, or cumulative workflow diagrams.

## A. ScrumBoard Taxonomy

Software engineering project courses typically involve multiple student teams, so each team in the ScrumBoard has their own *project*. A *project* has any number of members, each with their own *role*, e.g., guest, developer, leader, specific to that project. Each *project* has a *product backlog*, and can create any number of *sprints*. *User stories* are created in the *product backlog*, in preparation to be moved into a *sprint backlog* when they are ready. Each *user story* specifies *acceptance criteria*, and is broken down into *tasks*. Figure 1 depicts a partial sprint board with 2 stories and 9 tasks.

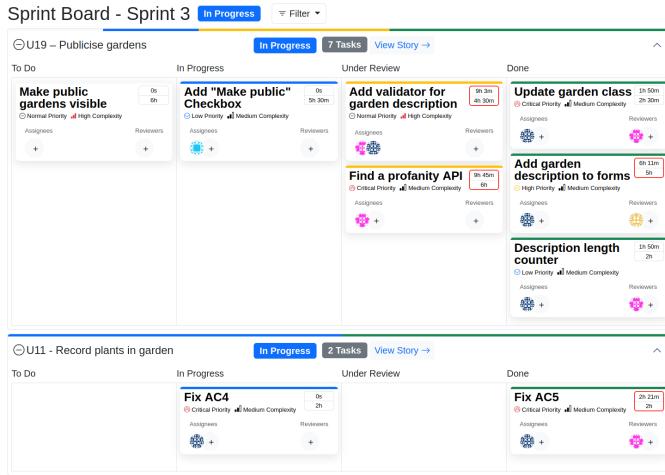


Fig. 1. Sprint board showing status of tasks and stories.

In addition to managing Scrum artefacts, the ScrumBoard supports many Scrum events. Daily Scrums can be scheduled for each *project*, which enables a UI notification in the sidebar counting down to the next Daily Scrum, and a dynamic iCal link that can be used to import Daily Scrums into all major calendar providers (e.g. Outlook, Gmail). Burn-down charts and cumulative flow diagrams are available for teams to inspect and adapt their work, as part of their Daily Scrums.

## B. Worklogs and Time Tracking

Students record and provide evidence for their work by creating *worklogs* for each *task* they work on, where each *worklog* can be linked to Git commits and *worklog tags*. *Worklog tags* allow students to break down how much of their time was spent on the following types of work for a given *worklog*: implementing features (including docstrings); writing documentation (e.g., design decisions, wireframes); creating automated tests or performing manual tests; fixing bugs; conducting code reviews; re-engineering or refactoring code; conducting spikes [9]. Additionally, students can log time spent in Scrum events, as well as all education specific sessions like technical workshops, or time spent writing their reflections or peer-evaluations, so all their effort spent in the course is recognised. Figure 2 depicts a summary of the work done by a team member during a sprint with the breakdown in stories, tasks, pair-programming, and review work accomplished.

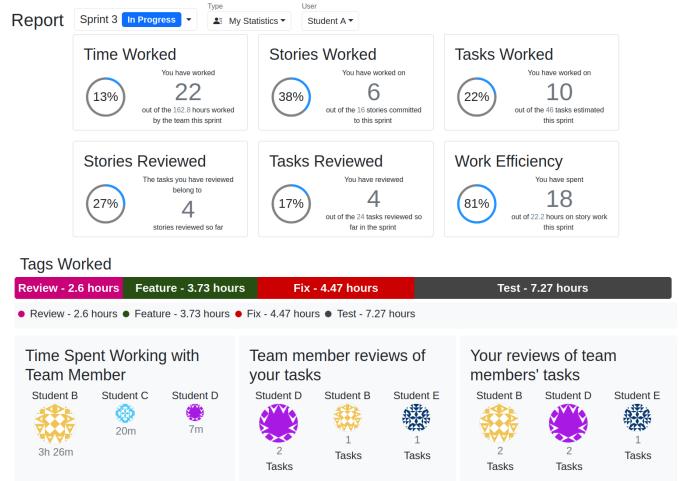


Fig. 2. ‘My Statistics’ view for a single sprint.

## C. Support for Sprint Reviews

When a sprint finishes it can be set as ‘Ready for review’ and teaching staff can assign ScrumBoard users, including members of another project to act as the reviewers and therefore conduct the Sprint Review, as a Product Owner would do. This feature guides reviewers through the review of each acceptance criterion for all non-deferred user stories of the sprint under review. The system supports concurrent editing and live-updates, so that any number of reviewers can work in parallel without issue, seeing changes from others as soon as they are made. The team under review can then view this Sprint Review at any time, and can see the assessment of each acceptance criterion for all stories, including rationale for their failure, when appropriate.

## D. Support for Reflective Practice

Software engineering project courses differ from industry software projects in that they include many practices that prioritise student learning or assessment over direct development of software artefacts, commonly [10]: self- and peer-evaluation questionnaires, or regular reflective journaling. Such practices are not directly supported by most industry standard tools, so teachers may rely on external systems like Google Forms. The ScrumBoard provides native mechanisms for enacting these practices within a course.

1) *Feedback Forms*: Teaching staff can create any number of *feedback forms* to send out to students. These are created through a similar interface to tools such as Google Forms, where teachers can create questionnaires with: blocks of text; multi-choice questions; and free-text questions. Forms can then be sent out to teams or individuals at different times and for different purposes, such as: student self-evaluation milestones; peer-evaluation questionnaires within teams; code reviews between teams; and, surveys.

2) *Weekly Reflections*: Enabling ‘Weekly Reflections’ for a project opens a weekly form for students to self-reflect, and a notification in the sidebar if the form has not been completed

for the current week. These forms encourage students to reflect on their learning journeys through simple questions, similar to a Scrum retrospective, but at the individual level: *What I did well this week; What I did not do so well this week; What I will do differently in future*; and optionally, *Anything else you want to let the teaching team know*. This approach allows students a great deal of freedom and flexibility in what they write, while still aligning with established theories of reflection such as Schön's concept of 'reflection-in-action' [11].

### E. Supporting Assessment

On top of team-based progress metrics, e.g., burn-down charts and cumulative flow diagrams, the ScrumBoard includes several visualisations only available to course teaching staff that can help educators to not only assess students more easily (see Section IV-A), but also to quickly identify and react to issues such as: individual students at risk of failing the course (Section IV-B); problematic student working patterns (Section IV-C); and, cliques, 'lone wolves', or knowledge 'silos' forming within teams (Section IV-D).

The 'Project Statistics' visualisation shown in Figure 3 displays the number of hours each student within a team has contributed to a given sprint, or for the project overall, in addition to the number of stories and tasks they have worked on or reviewed. This view allows teachers to quickly identify the students that may not be contributing much to a team, or dominating the workload of a team.

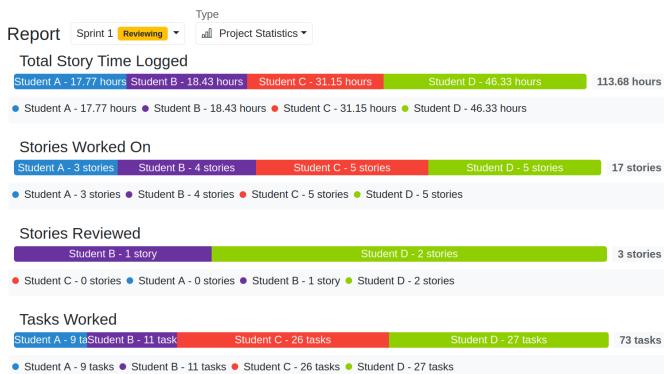


Fig. 3. 'Project Statistics' showing overview of team performance in a sprint.

The 'My Statistics' view shown in Figure 2 is available to all students for them to view information about their own work, but course teaching staff can view this information about any student. On this view you can see: the total time a student has spent working on tasks or in formal sessions (e.g., Scrum Events, workshops); the time spent against each worklog tag (see Section III-A); how much time they have spent pairing with each of their team-mates; how many tasks they have reviewed for each team member, and how many of their tasks have been reviewed by each team member; and, how many of the stories in the current Sprint Goal the student has worked on or reviewed. These stats can be viewed for any sprint in isolation, or for the project overall.

Metric	Week 19	Week 20	Week 21	Week 22	Total
Overhead	00:15:00	01:00:00	01:36:00	02:25:00	05:16:00
Story hours	01:03:00	09:56:00	09:40:00	09:02:00	1:05:41:00
Test hours	00:45:00	02:20:00	02:05:00	02:50:00	08:00:00
Average work log duration	00:12:36	00:29:48	00:30:32	00:39:30	
Shortest work log duration	00:01:00	00:01:00	00:01:00	00:05:00	
Testing contribution	88%	24%	25%	33%	29%

### Worklog History



Stories Worked	Total Effort	Total Time Spent
16	88	232h 24m

Date Occurred	Story Name	Task Name	Current Estimate	Time Spent	Description	Assignees	Worklog Tags	Issue Tags
05/10/2024 14:50:00	U10 - Edit Garden Details	T2 (AC6) Fix use comma for garden size.	1h 30m	1m	This task is the same as U8 T2, and was fixed when Anon. did the fix for U8 T1.	Anonym...	Fix	Missing commit
05/29/2024 01:11:00	U14 - Weather Monitoring	T4 (AC2) Get & display the weather for future.	5h	10m	Pair programming with Anon., fixed checking sunset sunrise times and added a default	Anonym... Anonym...	Fix	Outside work hours

Fig. 4. 'Marking Summary' with worklogs flagged by heuristics.

Finally, the 'Marking Summary' view shown in Figure 4 – only accessible by course teaching staff – contains a weekly breakdown for statistics such as: time spent in formal sessions; time spent working on tasks; time spent testing; and, average / minimum / maximum work duration. This visualisation uses a number of heuristics that highlight potential issues in student work sessions, such as: work completed outside advisable work hours (e.g., during the night); work logged without any Git commits linked as evidence; or sessions of work that overlap, potentially indicating a student that would artificially inflate their time spent on the project. These heuristics can quickly point to potential workload, engagement, or work ethics issues that teaching staff should explore further by looking deeper into a student's worklogs, or start a conversation with the student or team, when necessary.

### F. Additional Features

1) *Permission Levels and Roles:* The ScrumBoard comes with its roles and permissions that help manage users effectively. Next to a global admin role with all possible permissions, projects can have *leaders*, *developers*, *reviewers*, and *guests*. In short, leaders can edit all aspects of a project, including the members, and see the contribution of all developers. Developers can carry all activities to work on the project, but cannot see metrics about other developers apart from their worklogs. Reviewers can run a Sprint Review, but cannot access any other views. Guests can only see sprint boards and team-based progress metrics (i.e. burn-down charts and cumulative flow diagrams).

2) *Course Documentation:* A Git repository (currently GitLab specifically) containing markdown files can be linked to the ScrumBoard so that course handouts and other course documentation can be embedded for easy reference by students.

3) *User Identity Management*: The ScrumBoard can be used as a stand-alone tool where user accounts are created and managed internally, but also allows external identity sources, such as an organisation's active directory service.

#### IV. EXPERIENCES

We report our experiences using the ScrumBoard in a 3<sup>rd</sup> year, two-semester, software engineering group project with more than 200 students spread over 30 teams of 6-8 members since February 2022.

##### A. Supporting Assessment of Students

For each of the 6 to 7 sprints within the project course, typically lasting 3-4 weeks of in-term time, students receive formal feedback from the teaching staff on both their individual performance, and that of their team.

The process of marking students for each sprint requires teaching staff to inspect how well students follow Scrum, and apply its core values. To this end, teaching staff look at: students' worklogs completed on tasks and in formal sessions; their weekly reflections; the peer-feedback written by a student about their team-mates, *and* written by their team-mates about that student; and, the way students behaved in-person and communicated with their team. While direct observations about the latter are compiled by the teaching staff and gathered in a separate place, many of the process-related aspects of Scrum and agile software development (e.g., cross-functionality, automated testing, continuous delivery), as well as practical engagement in the course (i.e., timesheets) can be quantified with the data gathered from the ScrumBoard.

From prior observations, the total time required to mark a single student within a sprint, before the introduction of the ScrumBoard, was typically 1-3 hours, depending on the marker, the student, and team itself. Since the latest version of the ScrumBoard as depicted in this paper has been used in the course, the time taken to mark a student has now gone below one hour – often as low as 30-45 minutes. We attribute these time savings to two main reasons: firstly, the vast majority of data needed by assessors is now available from a single source which was designed to be used by assessors in the first place; and secondly, many of the manual calculations or investigations that assessors needed to perform (e.g., proportion of work for cross-functionality, engagement, missing evidence for worklogs) are now done automatically and shown directly to the assessor for review.

##### B. Early Interventions for Struggling Students

With the data and visualisations depicted in Figures 2 and 4, we have been able to identify struggling students much earlier in the course than we had in the past. Previously, teaching staff would have needed to extract data (manually or with purpose-built scripts) from a project management tool prior doing the marking at the end of a sprint, and then finally compile this as quantifiable evidence to identify issues with students. Now, the Scrumboard contains the latest and up-to-date snapshot

of a student's engagement with the course at all times, so a continuous monitoring of students' performance is possible.

While the course's organisation remained relatively stable over the last six years, this enhanced ability to observe the engagement of students in real-time led to a notable reduction in the number of students who withdrew from the course or received a failing grade overall, from an average of approximately 10% of withdrawal to less than 5% for the last instance of the course. We believe this is primarily due to early interventions with struggling students, enabled by the ScrumBoard's dedicated visualisation.

##### C. Identifying Problematic Work Patterns

Students frequently ignore assignments until very close to the deadlines, and will often work in large bursts outside of typical working hours (e.g. ‘pulling an all-nighter’). This kind of work hygiene is not representative of (most) industry development and is not sustainable, so should be discouraged.

To ensure that students practice working in a sustainable manner, and to be able to accurately assess them, we must be able to identify these problematic work patterns with tangible evidence. Such patterns could be identified by looking at timestamps of Git commits and other work artefacts, but this data is often insufficient, or takes too long to process. Also, it relies on students (or their peers) self-reporting these behaviors. The ScrumBoard provides heuristics in the ‘Marking Summary’ view (see Section III-E), which can automatically flag work done outside advisable working hours, so the teaching team can address these issues directly with students, and therefore prevent potential well-being issues.

##### D. Identifying Silos, Lone Wolves or Cliques within Team

Scrum is cross-functional by design, so it is crucial that students demonstrate their understanding of Scrum and, more generally, agile software development principles. In that regard, Scrum discourages specialisation in certain areas of the project by sticking to a small subset of the application's functionality; i.e., creating ‘silos’ of knowledge in the team.

Additionally, project courses that have bigger team sizes often run the risk of cliques forming within teams, or students that become ‘lone wolves’, preferring to work in isolation. It is often difficult to identify such cliques or isolated students without teaching staff witnessing something in-person or having it pointed out by other students through peer-evaluations.

Those issues can be identified with the visualisations described in Section III. As the ScrumBoard keeps a track of students working together as pairs when pair-programming a task, mappings of pair programming habits can be viewed on a sprint or whole-project basis. Therefore, the teaching staff can intervene during Daily Scrums to suggest particular team members to pair with each other to foster more knowledge sharing across the whole team. Previously, the lack of pair programming, or formation of cliques within teams required time consuming analysis of data extracts, so that they were formed from strong suspicions only, and required at least a full sprint before being identified and acted upon. Interventions can now be performed within the same sprint.

#### E. Tracing Students ‘Gaming’ the Marking

In addition to reducing the time required for marking students, we have also been able to provide a much greater level of detail to assessors about student actions, which has enabled us to more accurately identify instances where students attempt to ‘game’ the system, such as by retroactively inflating time spent in historic work sessions, abusing worklog tags to fake cross-functionality, e.g., claiming one piece of work as being both feature development and testing with no evidence.

The ScrumBoard keeps a complete, time-stamped, trace of all actions taken by users, both in terms of what pages they access in the system, but also a complete changelog of every entity a user creates or modifies, including exactly what changes were made. With this data we are able to produce a ‘replayable’ timeline of all actions performed by students which assessors can use to determine whether some ‘foul-play’ is occurring, e.g., students logging 30 minutes to complete their weekly reflection, but spending only a few minutes on the form, students retroactively increasing the time spent on work sessions weeks in the past, or on simple tasks with little code produced.

#### F. Centralised Access to Course Documentation

Another impactful feature we added to the ScrumBoard since its creation was to integrate a way for teachers to effectively host their own ‘wiki’ within the application, which centralises course documentation and makes it readily available for students. Our 3<sup>rd</sup> year project course has a very comprehensive document that clearly lays out expectations for students, regarding: their performance in the course; required deliverables within each sprint; how to reach out for help; and a variety of other resources and processes. Previously, this consisted of a single PDF document hosted on the course’s Moodle site, which was rarely looked at by students as it would typically take a few minutes to find and access, and was difficult to navigate within the document or find specific information. Moving this resource to the ScrumBoard as a series of searchable interlinking markdown pages significantly increased engagement with the documentation.

#### G. Students perception

We currently have not run dedicated surveys with students, but from anecdotal feedback, students have noted that the tool was complete but simple enough to introduce them to project management, e.g., “*It is clear and understandable enough for students to use efficiently (...) skipping the features that could make it complicated and overbearing*”, “*Compared to what I saw in industry (granted, extremely limited) although JIRA is more feature rich we didn’t really use anything that it provides which scrumboard does not.*”. Additionally, students mentioned that “*it was pretty much a single source of truth for the state of the project*”, and “*The metrics and graphs are a very useful feature for people to evaluate their own efficiency and alignment with agile practices*”.

#### V. CONCLUSION

In this paper we presented our experience with the ScrumBoard, a tool we developed to support project management, pedagogy, and assessment in software engineering project courses, all the while giving students experience that is directly transferable to the tools they will encounter in industry as they graduate. We reported our experiences using the ScrumBoard across several iterations of a 3<sup>rd</sup> year project course, and discussed how the ScrumBoard has significantly improved our experience as educators by streamlining assessment of students, identifying students’ engagement issues, and thereby allowing us to focus on course pedagogy.

In the future, we plan to add new features with additional heuristics to allow for identification of a more comprehensive set of problematic work patterns. We then plan to incorporate elements from the domain of intelligent tutoring systems, such as implementing ‘nudges’ for supporting student reflection through the ‘Weekly Reflections’, and a recommender system for helping students to assign themselves to appropriate tasks to help them widen their knowledge of the software they develop. Finally, we aim to integrate analysis of the code students link to their worklogs as evidence, both to ensure that they are accurately reporting their work, and to allow for automated identification of possible technical knowledge silos, or overuse of generative tools such as ChatGPT.

Alongside feature development, we plan to conduct studies to evaluate the effectiveness of the ScrumBoard as an educational tool. These will include analyses of quantitative usage data and student performance metrics, in addition to qualitative surveys to gather student perceptions of the tool.

#### REFERENCES

- [1] V. Mahnic, “A capstone course on agile software development using scrum,” *IEEE Transactions on Education*, vol. 55, no. 1, pp. 99–106, 2012.
- [2] S. Karunasekera and K. Bedse, “Preparing Software Engineering Graduates for an Industry Career,” in *20th Conference on Software Engineering Education & Training (CSEET’07)*, Jul. 2007, pp. 97–106.
- [3] P. Salza, P. Musmarra, and F. Ferrucci, *Agile Methodologies in Education: A Review*. Singapore: Springer Singapore, 2019, pp. 25–45.
- [4] M. Prince, “Does active learning work? a review of the research,” *Journal of Engineering Education*, vol. 93, no. 3, pp. 223–231, 2004.
- [5] P. Bourque, R. E. Fairley, and I. C. Society, *Guide to the Software Engineering Body of Knowledge (SWEBOK(R)): Version 3.0*, 3rd ed. Washington, DC, USA: IEEE Computer Society Press, 2014.
- [6] D. Draheim, “A cscw and project management tool for learning software engineering,” in *33rd Annual Frontiers in Education, 2003. FIE 2003.*, vol. 3, 2003, pp. S2C–31.
- [7] R. Q. Gonçalves and C. G. von Wangenheim, “How to teach the usage of project management tools in computer courses: A systematic literature review.” in *International Conference on Software Engineering and Knowledge Engineering*, 2015.
- [8] J. Lin, H. Yu, Z. Pan, Z. Shen, and L. Cui, “Towards data-driven software engineering skills assessment,” *International Journal of Crowd Science*, vol. 2, no. 2, pp. 123–135, Jan 2018.
- [9] W. Cunningham, “Spike solution,” online, accessed: 8 Apr. 2024. [Online]. Available: <http://wiki.c2.com/?SpikeSolution>
- [10] S. Tenhunen, T. Männistö, M. Luukainen, and P. Ihantola, “A systematic literature review of capstone courses in software engineering,” *Information and Software Technology*, vol. 159, p. 107191, Jul. 2023.
- [11] D. Schön, *The Reflective Practitioner: How Professionals Think in Action*. Ashgate, 1983.



# Evaluating Preschoolers' Block Programming Using Complexity and Personality Traits

Yui Ono

*Waseda University*

Tokyo, Japan

yui.yui.o@fuji.waseda.jp

Daisuke Saito

*Takachiho University*

Tokyo, Japan

d.saito@fuji.waseda.jp

Hironori Washizaki

*Waseda University*

Tokyo, Japan

washizaki@waseda.jp

**Abstract**—Programming learning at an early age effectively fosters logical thinking and self-centeredness, but an appropriate evaluation method for young learners has yet to be established. Herein we propose a new learning evaluation method that incorporates problem constructs and the complexity metrics used in software engineering quality assessments. Specifically, we investigate the relationships between changes in block programming complexity, personality traits, and learning effects. Evaluation rubrics and log data assess complexity, while personality traits are based on Big-5. Then the learning effects of 34 kindergarten children participating in workshops are analyzed in terms of complexity and personality traits. After learning, first-time programmers tend to show a large increase in complexity. The correlation with the rubric score is  $\rho = 0.43$ , and the correlation with log data is  $r = 0.92$ . Furthermore, analysis using the Big-5 gives  $\rho = 0.609$  for the rate of increase in extraversion and complexity, indicating a strong relationship between learning effects and personality traits. In the future, this data will be used to build AI tools for automatic evaluations, feedback, and learning curriculum recommendations.

**Index Terms**—Preschool programming education, Block programming, Complexity, Personality traits

## I. INTRODUCTION

Programming education should begin in kindergarten. LeChen et al. acknowledge the necessity for children to acquire computational thinking (CT) skills at an early stage [1]. According to Piaget's stages of development, the preoperational stage occurs in children ages 2-7 [2]. In this stage, children have yet to acquire the ability for mental operations. In other words, children at this stage are egocentric and assume that others share their point of view. Multiple approaches exist within the context of learning programming to achieve a solution rather than a single correct answer. This approach can be particularly effective for young individuals exhibiting egocentric tendencies, who may believe that their own answers are the only correct ones. Although learning programming at a young age can effectively an effective learning assessment for programming has yet to be established. One reason is that assessment methods, which evaluate students on the basis of performance, such as independent research in elementary school or graduation theses at university, are rarely implemented at the preschool level. Although Emily et al. developed a test to assess CT skills [3], few tools exist to evaluate students' learning processes. Establishing an evaluation method for young learners on the basis of students' exercises is necessary. Fronza

et al. focused on the complexity of Scratch programming and proposed a new complexity evaluation method using the "when" metric for block programming [4]. Although Fronza et al. found a positive correlation with Cyclomatic Complexity, a well-known text-based complexity, this metric may not be effective for programming beginners who struggle to handle branching processes. This study proposes a new performance evaluation method based on artifacts using complexity suitable for programming beginners, especially kindergarten students. Herein we apply the complexity metric used in software engineering evaluations to visual programming. Additionally, to promote individualized education [5], an analysis based on personality traits is conducted. This study examines our proposed evaluation method using Chuggington Programming, a block programming tool, and has three research questions.

- RQ1: How can the performance in visual programming be evaluated?
- RQ2: How does learning change the complexity of artifacts?
- RQ3: Are personality traits related to performance assessments?

Answering these questions makes two contributions. First, clarifying performance assessments by student artifacts should not only improve the evaluation of learning processes, but the use of complexity should measure the learning effectiveness quantitatively. Second, analyzing the relationship between personality traits and learning effectiveness should provide generous support tailored to students' characteristics, allowing them to fully develop their abilities. The rest of this paper is organized as follows. Section 2 describes the background of the study. This includes previous studies measuring the complexity of visual programming, the personality traits used in the analysis, and the visual programming tool used in this study, Chuggington Programming. Section 3 details the performance evaluation method and the content of the workshops. Section 4 describes our results. Section 5 discusses the RQs, threats to validity, and future perspectives. Section 6 reviews related studies. Finally, Section 7 presents the conclusions.

## II. BACKGROUND

### A. Complexity of programing

Halstead proposed metrics to measure the complexity of software. These operators are determined by four variables [6]:

$$\begin{aligned}\eta_1 &= \text{Number of distinct operators} \\ \eta_2 &= \text{Number of distinct operands} \\ N_1 &= \text{Total number of operators} \\ N_2 &= \text{Total number of operands}\end{aligned}$$

These operators are defined by the equations below. Equation (1) represents D and indicates the level of difficulty of the program. Equation (2) represents V and measures the minimum number of bits required for coding the program. Equation (3) represents E and assesses the intellectual effort required to create or maintain the program.

$$D = \frac{\eta_1}{2} \times \frac{N_2}{\eta_2} \quad (1)$$

$$V = (N_1 + N_2) \log_2(\eta_1 + \eta_2) \quad (2)$$

$$E = D \times V \quad (3)$$

### B. Based on Big-5 Personality

Samuel et al. developed a shortened measurement tool called TIPI [7]. TIPI is a simplified method to assess the personality traits on the basis of Big-5[8]. TIPI is comprised of ten items. Each item is evaluated on a scale ranging from 1 (strongly disagree) to 7 (strongly agree). The score is used to derive an indicator of five traits: extraversion, agreeableness, openness to experience, neuroticism, and conscientiousness. Additionally, Koishio et al. created a Japanese version of TIPI-J and demonstrated that observable traits moderately align between peer and self-assessment [9].

### C. Chuggington Programming

Chuggington Programming targets individuals ages 3 and older [10] and fosters programming-oriented thinking. Users navigate trains from the starting point to the goal by combining block codes. Fig.1 illustrates an example of Chuggington Programming in answer mode. Users not only solve pre-set courses, but they can also create their own original challenges (Fig. 2). Sato emphasized the effectiveness of problem-based learning in enhancing understanding within programming education [11]. This study utilizes Chuggington Programming for two reasons. First, it has a lower target age range compared to other visual programming tools like Scratch or Blockly. Second, it supports problem creation.



Fig. 1. Course-solving Mode



Fig. 2. Course-creating Mode

## III. METHOD

Workshops were held at an educational institution to validate our evaluation method using complexity. This chapter details the workshops and the proposed performance evaluation of block programming.

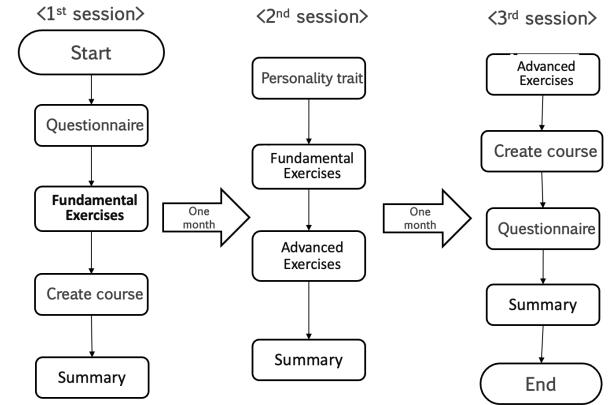


Fig. 3. Workshop Flowchart

### A. Workshop

Monthly workshops were held at a licensed kindergarten for three months. Each session was 90 minutes. The participants were 34 kindergarten students ages 4–5 years old. The workshops have the following objectives:

- Nurture an interest in programming
- Highlight that computers follow procedures
- Learn to follow procedures through programming
- Reinforce that there are multiple ways of thinking in programming

This workshop had two modes: course-solving mode and course-creation mode. We analyzed the changes in the students' scores and the complexity of the original course programs created by the students before and after the study. Fig.3 overviews the workshop flow. In the course-solving mode, the basic rules were explained, and then students were free to choose a course and deepen their understanding of the algorithm. In the course-creation mode, students were given 15 minutes of practice time to create their own original course. The collected data can be divided into four categories: (1) basic information such as student gender, age, and programming experience, (2) original course programs before and after the workshop, (3) personality characteristics, and (4) log data during the workshop. TIPI-J was used for the personality characteristics[8]. Since the participants were between 4 and 5 years old and it was difficult for them to answer the questions themselves, the personality characteristic data were provided by three teachers who knew the participants well.

### B. Performance evaluation

The original courses created by the students were analyzed using two methods: scoring based on a rubric and measuring complexity using turtle graphics.

1) *Rubric score*: Relevant criteria were selected on the basis of the Rubric ProEEs proposed by Saito et al. [12] and scored for this workshop series. Table 1 lists the four criteria used in this study. C2 indicates understanding of programming concepts, while C3 denotes understanding of

TABLE I  
RUBRIC SCORE BASED ON RUBRIC PROEES

Code	Description	Detailed Explanation
C2	Sequential Execution	Able to move the train
C2	Iteration	Iteration: Able to use loops
C2	Conditional Branching	Able to use conditional statements
C3	Understanding of Computer Principles	Able to set up the start and goal
C3	Understanding of Computer Principles	Able to create a functional course
C4	Creativity	Able to decorate
C4	Expressiveness	Able to set up bridges, buttons, level crossings, etc
C5	Creation of Programs Using Each Element	Able to write programs for created courses
C5	Logical Thinking in Program Creation	Able to write programs for the shortest path

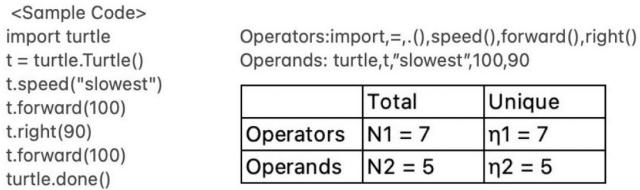


Fig. 4. Visual Program Complexity Measures

computer principles. C4 represents designing programs, and C5 evaluates creating programs.

2) *Complexity*: We measured the complexity of a set of visual programming block codes using the method proposed by Ono et al. [13]. The course programs written in block-based code and created by the students before and after the study were converted into text-based code using turtle graphics. Table 2 shows the code conversion rules. Then the complexity was measured using a library called multimetrics. A score of 0 indicates that a course was not established. Ono et al. used four metrics to assess program complexity: Cyclomatic Complexity, Halstead Difficulty, Halstead Volume, and Halstead Effort. Their study noted a strong correlation between the Halstead Difficulty and the perceived difficulty of visual programming [13]. We did select Halstead Difficulty because this metric analyzes the difficulty level of the programs. Figure 4 shows an example of the visual programming complexity measurement shown in Fig. 1. Substituting into equation (1), we obtained D = 3.5.

3) *Logdata*: Log data is the data from the Chuggington Programming solution mode. It consists of course completion counts, execution counts, and response times. We also added the complexity of the block codes created by the students in solution mode to the log data score.

#### IV. RESULT

##### A. Relationship between Rubric Score and Complexity

Table 3 presents the results of the Rubric ProEEs scores based on the first and third sessions as well as the Halstead Difficulty scores. Fig.5 illustrates a histogram of the rubric scores, while Fig.6 depicts the complexity scores based on the level of experience. The mean rubric score increased by 0.97 points from 1.39 to 2.36 (Table 3). The standard deviation became smaller for both rubric scores and complexity exhibited in the third session, indicating decreased data dispersion following programming instruction. The Shapiro-Wilk test for normality revealed that all p-values were below 0.05, except

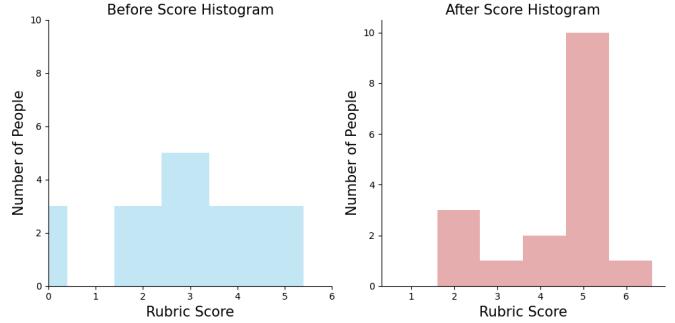


Fig. 5. Comparison of Rubric Scores

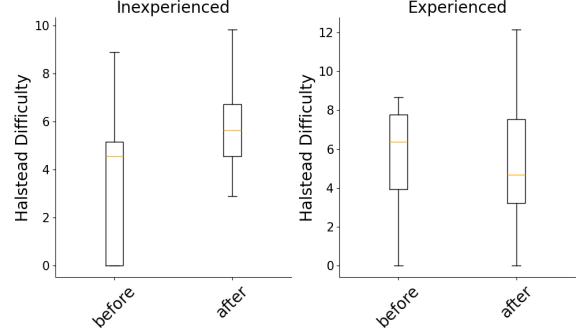


Fig. 6. Comparison of Halstead Difficulty

for Complexity (After)[14]. Since the results deviated from normality, a Wilcoxon signed-rank test was conducted as a non-parametric paired comparison between pre- and post-assessments[15]. The null hypothesis was that there is no difference between pre- and post-assessments. The alternative hypothesis was that there is a difference between them at a significance level of  $\alpha = 0.05$ . The results showed a significant difference in rubric scores ( $p = 0.0020 < 0.05$ ), supporting the alternative hypothesis. The mode before instruction 3 shifted to 5 after instruction, suggesting that the workshop facilitated the acquisition of algorithmic and CT skills. Conversely, the p-value for complexity was 0.21 ( $p < 0.05$ ) and failed to reject the null hypothesis. Fig.5 may reveal the reason. The increase in complexity was greater for the inexperienced students and smaller for the experienced ones. Considering the lack of adherence to statistical normality indicated by Table 3, Spearman's rank correlation coefficient was computed to assess the relationship between rubric score and complexity[16]. The moderately strong correlation ( $\rho = 0.43$ ) suggests the potential utility of complexity as an evaluation metric.

##### B. Relationship between Log Data and Complexity

The relationship between log data and complexity was analyzed. Pearson's correlation coefficient was calculated since both the log data and complexity scores were on a ratio scale. The strong correlation ( $r = 0.92$ ) suggests a robust association between the number of problems solved by students and the learning effectiveness assessed by the performance evaluation.

TABLE II  
CONVERSION RULES TO TURTLE GRAPHICS

Chuggington	Turtle graphics	Details
Wilson (Red train)	t.color('red')	Specify train type
Koko (Green train)	t.color('green')	Specify train type
Brewster (Blue Train)	t.color('blue')	Specify train type
Go forward one square	t.forward(100)	Advance the train one square in the direction of travel
Go back one square	t.back(100)	Move the train one square in the opposite direction of travel
Turn right	t.right(90)	Change the direction of the train 90 degrees clockwise
Turn left	t.left(90)	Change the direction of the train 90 degrees counterclockwise
Lift up	t.color('pink')	Raise the lift one level
Put down	t.color('lightblue')	Lower the lift by one step
Press the button	t.color('yellow')	Push the button to move the device

TABLE III  
STATISTICS OF RUBRIC AND COMPLEXITY MEASURES

	Mean	Standard Deviation	p-Value
Rubric (Before)	2.83	1.57	0.04
Rubric (After)	4.22	1.23	0.001
Complexity (Before)	5.08	3.20	0.02
Complexity (After)	6.05	2.39	0.12

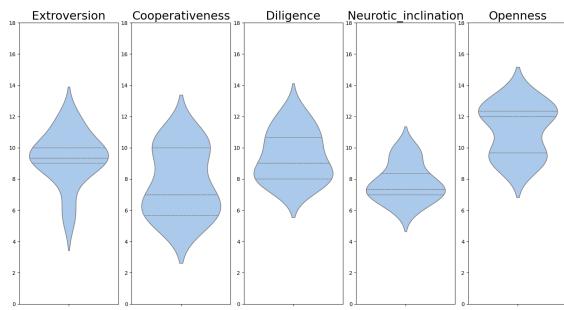


Fig. 7. Violin Plot of Personality Traits

### C. Relationship between Personal Traits and Complexity

Scores for the five personality traits were calculated on the basis of TIPI-J. Fig.7 shows a violin plot of the distribution of the students' personality traits. The wavy lines in the violin plot represent quartiles. Openness had a slightly higher overall score, while neuroticism had a smaller variance. Next, we examined the relationship between personality traits and complexity. Personality traits were on an ordinal scale, whereas complexity was on a proportional scale. Therefore, we used Spearman's rank correlation coefficient. Table 4 shows the results of the pre-complexity, post-complexity, and their differences as well as the correlation coefficients of the five personality traits. There were strong correlations between openness, diligence, and pre-complexity. In addition, the post-complexity showed positive correlations between personality traits and all factors. Moreover, extraversion was strongly correlated with the difference between pre-complexity and post-complexity and personality traits.

## V. DISCUSSION

### A. Responses to Research Questions

1) *RQ1: How can the performance in visual programming be evaluated?*

TABLE IV  
COMPARISON OF FIVE-ITEM PERSONALITY INVENTORY AND COMPLEXITY

Five-Item Personality	Complexity (Before)	Complexity (After)	Difference
Extraversion	-0.261	0.382	0.609
Agreeableness	0.500	0.527	-0.151
Conscientiousness	0.550	0.436	-0.411
Emotional Stability	-0.201	0.189	0.269
Openness to Experience	0.671	0.551	-0.171

: In this study, we proposed a method to evaluate performance on the basis of artifacts by assessing how the algorithm creation skills change after learning the basic algorithms necessary for programming in a series of three workshops. Since self-assessments of kindergarten students are challenging, evaluations tend to be based on results such as quizzes or tests. Consequently, incorporating performance evaluations of the learning process into programming education of kindergarten students holds promise as a new method to measure learning outcomes.

2) *RQ2: How does learning change the complexity of artifacts?*

: Here, we proposed a new evaluation method focusing on the complexity of artifacts and discussed its relationship with a conventional rubric evaluation and log data. The average Halstead Difficulty increased before and after the class. However, after the class, the standard deviation decreased and the distribution became narrower. Typically, lower values of software complexity are better. Since this study targeted programming beginners, most students found it difficult to write systematic programs. Complexity increased significantly for inexperienced learners (Fig.6), while that for relatively more experienced individuals showed a negligible change, but the average values decreased slightly. This suggests that among students becoming accustomed to programming, those who write more compact and systematic programs increase. Therefore, understanding algorithms in the early stages of learning and difficulty programming are directly related and may serve as indicators of learning outcomes for beginners.

3) *RQ3: Are personality traits related to performance assessments?*

: We analyzed the relationship between personality traits

on the basis of TIPI-J and complexity using Spearman's rank correlation coefficient. Strong correlations were observed between pre-learning complexity and openness. Openness is an indicator of adventurousness, curiosity, and acceptance of new experiences. Therefore, it is believed that openness and complexity show a strong correlation among programming beginners. Additionally, the difference in complexity before and after learning was strongly correlated with extraversion. Extraversion is an indicator of vigor and assertiveness, implying that students who communicate with others and incorporate new ideas show higher growth rates. These results suggest that personality traits are related to learning outcomes. Since previous studies have not analyzed the relationship between kindergarten learning outcomes and personality traits, further analysis of these relationships may lead to a more comprehensive learning support.

#### B. Threats to Validity

There are two threats to validity. First, the data is limited. The participants were kindergarten students, and the proportion of students who attended all three workshops was low. Second, the use of the TIPI-J for test simplification categorized personalities into five traits on the basis of teachers' input. However, the results may differ if the responses were obtained by individuals who knew the students better such as a family member.

#### C. Future work

Future efforts will focus on increasing the number of data points and expanding the analysis beyond personality traits. For example, we would like to incorporate text mining based on free responses. Additionally, we are developing an AI programming learning support tool targeting kindergarten students. This software aims to provide efficient learning support to students by offering features such as automatic grading and personalized learning course recommendations based on individual learning styles. The ultimate goal is to reduce the workload for teachers while enhancing the effectiveness of student learning.

#### VI. RELATED WORK

Chou et al. investigated the development of young children's computational thinking (CT) skills using ScratchJr through pre- and post-tests and qualitative observations [17]. While many studies implement pre- and post-tests, our study's novelty lies in evaluating students' learning processes using performance assessments and detailed analyses involving personality traits. In another study, Pila et al. used Daisy the Dinosaur and Kodable to assess young children's programming learning outcomes [18]. They evaluated understanding through commands like "when," "move," "grow," and "jump." Algorithmic knowledge but not verbal expression of coding concepts improved in-game. The lack of improvement in verbal expressions was attributed to young children's verbal articulation challenges. This highlighted the need for alternative assessment methods. Our proposed composition-based assessment directly evaluates performance and bypasses the need for verbal articulation.

#### VII. CONCLUSION

This study proposed performance evaluation method is proposed and its correlation with personality traits is analyzed. The use of complexity seems to be an effective performance evaluation method for programming beginners. Moreover, students with higher levels of openness showed strong correlations with pre-assessment, while those with higher levels of extraversion exhibited higher growth rates. Therefore, analysis that includes personality traits may realize personalized learning and the development of more effective AI tools to support learning.

#### ACKNOWLEDGMENT

We would like to express our deep gratitude to Kids Garden, Fuji Television Network, Inc., and KOYU BUSINESS CO., LTD for their generous cooperation and support in the execution of this research.

#### REFERENCES

- [1] Z. LeChen, and N. Jalal, "A systematic review of learning computational thinking through Scratch in K-9", *Computers & Education*, Vol. 141, 2019.
- [2] S. Osawa, "The Concept of Cognitive Development in Piaget's and Vygotsky's Theories: Implications for Language Acquisition Research", *Hiroshima Shudai Ronshu*, Vol. 49, No. 2, pp. 1-11, 2008.
- [3] E. Relkin, L. Ruiter, and M. Bers, "TechCheck: Development and Validation of an Unplugged Assessment of Computational Thinking in Early Childhood Education", *Journal of Science Education and Technology*, Vol. 29, 2020.
- [4] I. Fronza, L. Corral, and C. Pahl, "An Approach to Evaluate the Complexity of Block-Based Software Product", *Informatics in Education*, Vol. 19, No. 1, pp. 15–32, 2020.
- [5] H. Yoshida, "Focusing on "Preparing for Future Learning" Research", *Transactions of the Japan Society for Educational Technology*, Vol. 46, pp. 393-403, 2022.
- [6] H. Halstead, "Elements of Software Science", Elsevier North-Holland, New York, 1977.
- [7] S. D. Gosling, P. J. Rentfrow, and W. B. Swann, "A very brief measure of the Big-Five personality domains", *Journal of Research in Personality*, Vol. 37, pp. 504–528, 2003.
- [8] L. R. Goldberg, "Language and Individual Differences", *Review of Personality and Social Psychology*, Vol. 2, pp. 141-165, 1981.
- [9] S. Koshio, S. Abe, and P. Cutrone, "Attempt to Create the Japanese Version of the Ten Item Personality Inventory (TIPI-J)", *Personality Studies*, Vol. 21, No. 1, pp. 40-52, 2012.
- [10] Fuji Television Network, "Chuggington", <https://www.fujitv.co.jp/chuggington/>
- [11] H. Sato, "Quantitative Evaluation of Learning Improvement through Problem-Based Learning in Programming Education", 2019.
- [12] D. Saito, A. Sasaki, H. Washizaki, Y. Fukazawa, Y. Muto, M. Tamura, and T. Nishizawa, "Proposal for Rubric for Elementary School Programming Education", *STEM Education Research*, No. 1, pp. 41-51, 2018.
- [13] Y. Ono, D. Saito, H. Washizaki, and Y. Fukazawa, "Measuring Complexity in Visual Programming for Elementary School Students", Vol. 32, pp. 103-112, 2024.
- [14] S. S. Shapiro, and M. B. Wilk, "An Analysis of Variance Test for Normality (Complete Samples)", *Biometrika*, Vol. 52, No. 3/4, pp. 591-611, 1965.
- [15] F. Wilcoxon, "Individual Comparisons by Ranking Methods", *International Biometric Society*, Vol. 1, No. 6, pp. 80-83, 1945.
- [16] C. Spearman, "The Proof and Measurement of Association between Two Things", *The American Journal of Psychology*, Vol. 15, No. 1, pp. 72–101, 1904.
- [17] P. Chou, "Using ScratchJr to Foster Young Children's Computational Thinking Competence: A Case Study in a Third-Grade Computer Class", *Journal of Educational Computing Research*, Vol. 58, pp. 570-595, 2020.
- [18] S. Pila, F. Aladé, K. Sheehan, A. Lauricella, and E. Wartella, "Learning to code via tablet applications: An evaluation of Daisy the Dinosaur and Kodable as learning tools for young children", *Computers & Education*, Vol. 128, pp. 52-62, 2019.



# Exploration and Practice of University-Enterprise Deep Integration Oriented Talent Training Approach for Software Engineering Specialty

Shuwei Qiu, Wei Huang, Gengzhong Zheng, Xiaojia Zhou, Mahammad Humayoo<sup>✉</sup>  
Hanshan Normal University, Chaozhou, Guangdong 521041, P. R. China  
humayoo@hstc.edu.cn

**Abstract**—A University-Enterprise Deep Integration-Oriented Talent Training Approach (UED-IOTTA) was designed to improve the post-competency and employment prospects of software engineering graduates from colleges and universities. This approach is tightly aligned with the demand for talent in enterprises and the development of talent in academic institutions on a variety of dimensions, including employment requirements, professional ethics, and occupational skills. Building on this paradigm, a specific implementation strategy has been developed. This implementation strategy's key components include establishing industry-specific training classes known as UED-IOTTA classes, providing software engineering vocational training, putting industry-academic collaboration supervision techniques into practice, developing progressive pedagogy, and defining particular university-enterprise partnership strategies. Empirical results indicate the noteworthy influence of this customized talent training approach on improving the caliber of talent advancement in the field of software engineering.

**Index Terms**—Software engineering, oriented talent training, university-enterprise integration, industry-university-research

## I. INTRODUCTION

Over the past few years, numerous higher education institutions have incorporated engineering disciplines like software engineering, artificial intelligence, and cloud computing into their curriculum. This has resulted in a significant influx of students from various backgrounds who are now pursuing majors in these disciplines [1]. Software engineering stands out as the field with a significant number of graduates. As a result, there is fierce rivalry among software engineering graduates for job opportunities. Furthermore, the present employment rate and prospects are suboptimal [2]. There are still several obstacles that remain in the hiring of software engineering graduates: (i) Insufficient Post-Competency: Post-competency refers to the skills, knowledge, and abilities that an individual needs to possess in a specific position, as well as the ability to effectively apply these skills, knowledge, and abilities in their work. Software engineering necessitates interdisciplinary knowledge, hence requiring students to acquire robust comprehensive abilities across several areas. Nevertheless, acquiring proficiency in a wide range of skills can prove to be

This work was supported by the Natural Science Foundation of Guangdong Province under Grant 2022A1515010990, Innovation Teams of Ordinary Universities in Guangdong Province under Grants 2023KCXTD022 and 2021KCXTD038, and Key Laboratory of Ordinary Universities in Guangdong Province under Grant 2022KSYS003.

difficult, leading to a lack of adequate expertise for certain graduates. (ii) Limited Continuous Learning Ability: The fast-paced software engineering industry and related technologies expect graduates to be strong and adaptive learners. Traditional teaching methods may not develop this skill. (iii) Insufficient Practical Experience: Employers frequently look for applicants with practical experience because software engineering is a practically oriented field. Some graduates struggle to quickly adjust to the demands of the industry because they were not exposed to real-world situations or given opportunity to gain experience while still in university.

In response to the issues outlined earlier, a talent training focused on job market skills has emerged [3]. This approach is customized to cater to the precise requirements of various sectors or jobs, providing numerous benefits: (i) Enhanced Matching: This approach connects training objectives closely with the requirements of specific roles, guaranteeing that the developed skills are more suitable to satisfy the unique demands of the job market/position. Consequently, this improves the employability of graduates. (ii) Focus on Practicality: With a priority on practical instruction and training, this approach emphasizes the development of students' practical skills and problem-solving capabilities, preparing them to effectively tackle real-world issues they may encounter in their respective fields. (iii) Enhanced Efficiency: By prioritizing the development of job-specific talents and skills, this approach closely aligns teaching content with industry demands, resulting in improved efficacy and efficiency in nurturing talent. (iv) Increased Career Opportunities: By fostering collaboration and active involvement with businesses, this approach offers students a wide range of potential career growth opportunities. Internship training and employment advice provide students with opportunities to quickly secure relevant jobs.

This research introduces a UED-IOTTA for the software engineering programme at our university, taking into account the advantages provided by the talent training approach mentioned in the study by [4]. This paradigm is based on multiple dimensions, including employment requirements, professional ethics, and occupational skill demands. It effectively combines the need for talented individuals from enterprises with the educational curriculum offered by colleges and universities. The goal is to enhance graduates' employability and post-competency.

## II. UNIVERSITY-ENTERPRISE DEEP INTEGRATION ORIENTED TALENT TRAINING APPROACH

The UED-IOTTA, as presented in this paper, is depicted in Figure 1. The diagram illustrates the integration between enterprises and universities across seven dimensions: “Job Requirements-Talent Training Objectives”, “Professional Ethics-Public Course System”, “Occupational skill Requirements-Professional Curriculum System”, “Skill Level Requirements-Evaluation System”, “Tasks or Projects-Internship Training”, “Task or Project Acceptance-Course Assessment” and “Graduate Employment-Graduation of University Students”. The downward vertical arrows in the figure represent the progressive relationship between these dimensions, while the horizontal two-way arrows symbolize the integration and cooperation between enterprises and universities within each dimension, highlighting the deep integration between the two entities.

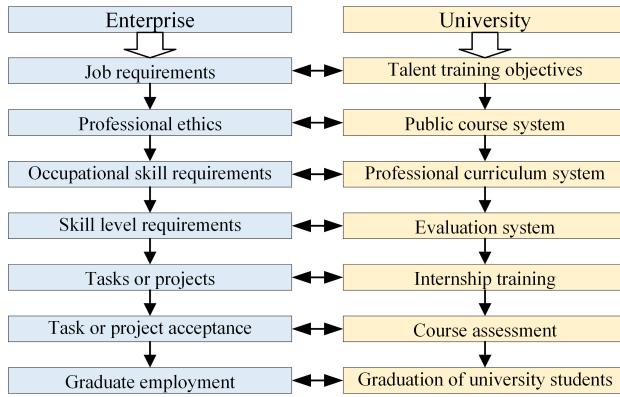


Fig. 1. Deep integration oriented talent training approach.

The Fig.1 illustrates the comprehensive integration of enterprises and universities, which aims to achieve overarching goals and strategic alignment. This integration includes personnel training, research collaboration, joint development of internship training facilities, and sharing of information resources [5]. By partnering with universities and colleges, enterprises can improve the general caliber and creative mindset of their employees, gain access to cutting-edge technological research discoveries, decrease expenses related to product research and development, and efficiently and effectively attract top-tier talent. Simultaneously, colleges and universities can form partnerships with enterprises to get insight into their job prerequisites and employment trends, refine talent training programs, and raise the caliber of talent cultivation. This, in turn, will enhance the job proficiency and employment prospects of graduates. The close collaboration between enterprises and academic institutions promotes the synergistic development of Industry-University-Research integration, which in turn nurtures the growth of top-notch talent for the betterment of society.

The primary principles of the talent development approach described in this research are summarized as follows:

### A. Integration of job requirements and talent training goals

A strong foundation for developing talent in line with enterprise demands can be established by carefully analyzing enterprise job requirements and then customizing talent training objectives to satisfy these needs. In order to properly connect course material and teaching techniques with real employment requirements, schools and universities must collaborate with enterprise to develop talent training objectives. This collaboration provides talent assistance for the enterprise.

### B. Integration of professional ethics and public course system

Public courses are fundamental to college education, with the goal of developing students' overall abilities, basic skills, and professional ethics. The integration of professional ethics into public courses entails the creation of focused teaching plans that include specific content on professional ethics. These plans are designed to be incorporated into courses that cover topics such as legal fundamentals, career planning, and innovation and entrepreneurship. In addition, the act of inviting enterprise lectures on professional ethics and requirements offers students actual opportunities to develop professional ethics, so improving their professional competency and social accountability.

### C. Professional curriculum and occupational skill requirements integration

When creating the professional curriculum, it is crucial to thoroughly take into account the distinct features of the software engineering industry and the skill prerequisites that enterprises need. Specifically, one can select courses that concentrate on software project management, Capability Maturity Model Integration (CMMI), software architecture, and software engineering to match the skill demands commonly found in the software engineering industry. These courses seek to provide students with extensive knowledge and skills in designing, developing, implementing, and maintaining software systems, consequently improving their proficiency in the field of software engineering. The creation and design of the professional curriculum outline, which involves determining curriculum objectives, developing teaching content, and assessing instructional progression, requires active participation from both technical personnel and management staff inside the company. This collaboration promotes a more profound comprehension of industry requirements and enables the smooth incorporation of practical skills into the curriculum. By matching project objectives and occupational skill requirements with the professional curriculum system, a direct correlation can be formed. This linkage promotes the reconstruction of the curriculum system and the development of instructional content that is specifically suited to meet the needs of the students and enterprises.

### D. Skill level requirements and evaluation integration

The course evaluation system should develop appropriate criteria and methodologies that align with the skill level requirements of the industry in order to impartially evaluate

the proficiency of students. Course evaluation measures the extent to which students are acquiring the skills required by enterprises through the training courses they are completing. If this is the case, it also assesses their proficiency level in that specific ability at different intervals during the training programme. Consequently, regularly monitoring and adapting teaching tactics throughout the teaching process guarantee consistent enhancement of abilities, promoting innovation and collaboration among students, and equipping them with the necessary employment skills.

#### *E. Integration of tasks or projects and internship training*

It is essential to incorporate internship training for interns with work tasks or real-world projects from enterprises. Students obtain a deeper grasp of the real-world workflow, project development, and implementation processes in the enterprises through internship programmes. This allows them to obtain a greater understanding of the practical application of software engineering, while enterprises have the chance to assess and hire students as workers throughout the internship time, allowing them to hire the skilled individuals required for business growth. They can create a thorough strategy for internship training together and take an active role in managing and assessing internship programmes.

#### *F. Task/project acceptance and course assessment integration*

When a student enrolls in the UED-IOTTA, the partner company assigns a real-time project called the industry-standard skill assessment project to evaluate the student's skill. Students work together with professionals from the partner company to take on and finish this assignment. After the project is finished, professionals assess it to make sure it satisfies industry standards. If the project satisfies these requirements, it means that the industry-standard skill assessment and course requirements were successfully completed, verifying the knowledge gained from training courses. If not, the project is turned down, demonstrating a lack of understanding from the relevant training course.

Furthermore, the incorporation of work tasks or project acceptance inside course assessment is crucial for promoting the integration of production and education, consequently improving the quality of teaching. By integrating the acceptance criteria and processes of software engineering projects into course assessments, teachers can acquire more profound insights into crucial teaching points and challenges. This allows them to effectively mentor students in comprehending the practical execution and acceptance prerequisites of software engineering projects. At the same time, businesses that take part in course evaluations receive vital information on the caliber of teaching and training provided by schools and universities. This promotes greater collaboration and interaction between educational institutions and enterprises, thereby significantly improving and advancing the teaching efficacy of the software engineering field. This joint strategy ensures curriculum evaluations meet industry standards and help students and enterprises succeed.

#### *G. University student graduation and graduate employment*

The assimilation of newly hired employees and college students' graduations is a critical connection in the development of proficient individuals. By collaborating with colleges and universities, enterprises may include their requirements, expertise, and job skills into talent training programmes, allowing college students to better meet industry demands and adapt to the workplace. Simultaneously, internship opportunities provided by enterprises enable students to obtain a deeper understanding of workflow and job needs, laying the groundwork for a smooth transfer into the workforce after graduation. This integration not only makes it easier for enterprises to find qualified employees, but it also enables college graduates to find work and advance their careers, resulting in a win-win scenario.

Furthermore, university-enterprise cooperation provides students and teachers with opportunities to acquire new knowledge. This collaborative method enables them to "go out" to enterprises, gaining vital insights and expertise, while enterprises "come in" to universities, employing university resources to obtain new research findings. Such collaborative research at universities benefits both sides.

Ultimately, it is crucial to promptly execute the UED-IOTTA in order to fulfil the requirements of the job market for graduate students.

### III. IMPLEMENTATION OF THE TALENT TRAINING APPROACH

In this section, we outline the specific implementation of the proposed UED-IOTTA, focusing on overall objectives, methods, measures, and progress planning.

#### *A. Overall objectives*

Xinyingda Technology Co., Ltd., our enterprise partner, is essential to the UED-IOTTA. A supply-demand docking-oriented talent training programme is launched by our university's School of Computer and Information Engineering in collaboration with our enterprise partner [6]. This programme aims to smoothly incorporate enterprise employment requirements into the software engineering specialization's talent training system, making it easier for graduates to land high-paying jobs and improving their post-competency. This will enable us to supply the software engineering industry with top-notch human resources and a skilled workforce for software engineering projects. Ultimately, this alliance aims to create a approach of university-enterprise cooperation [7], which promotes mutual growth and synergy between industries and disciplines.

#### *B. Approaches and measures*

The approaches and measures are explained as follows: Our university has begun the application process for funding through the Ministry of Education's supply and demand docking employment education project, in collaboration with Xinyingda Technology Co., Ltd. The UED-IOTTA is successfully implemented through the execution of this project. The precise measures are outlined below:

TABLE I  
TIMELINE AND CORRESPONDING IMPLEMENTATION TASKS

Time (months)	Implementation Tasks
01 to 03	Establish an industry-specific training class titled “Xinyingda Software Engineering” tailored to meet the company’s employment requirements. Formulate a comprehensive talent training plan and curriculum outline through university-enterprise integration.
04 to 06	Representatives from the university’s software engineering department engage in communication and discussion with relevant management personnel and technical experts from the company to formulate a specific training plan for the University-Enterprise Deep Integration Oriented Talent Training.
07 to 12	Our university collaborates with the company on three categories of training courses, utilizing software engineering methods and processes: embedded software development, artificial intelligence, and intelligent hardware. These training programs primarily consist of software engineering training camps, online lectures, or on-site seminars, with engineers from the company serving as lecturers. Throughout the implementation process, students and faculty members of this major work alongside the company to develop relevant teaching resources, enhancing the quality of instruction and fostering technological innovation.
13 to 20	Arranging internships and practical training for students enrolled in the industry-specific training class, linking them with relevant companies. These companies offer students hands-on training opportunities aligned with industry job requirements. Students in the industry-specific training class collaborate with the companies to undertake and execute real software engineering projects, gaining valuable practical experience. Throughout this process, professional teachers and company representatives collaborate to guide students in engaging with innovation and entrepreneurial practices.
21 to 24	The enterprise and our university collaboratively review and evaluate students’ work tasks or projects, drafting project implementation reports and analyzing teaching outcomes. Exceptional graduates whose project assessments meet the criteria are given priority in recruitment by the enterprises.

*1) Industry-Specific Training Class:* We establish an industry-specific training class named “Xinyingda Software Engineering” within the software engineering major at our university. Leveraging the technical expertise of the enterprise and aligning with industry-specific talent demands, we scientifically and thoughtfully design the oriented talent training program. Additionally, we implement the university-enterprise “double tutor” system [8], wherein each student is assigned both an on-campus mentor and a corporate mentor. This system ensures regular assessment of students’ progress and maintains teaching quality. The industry-specific training class spans 24 months, covering the junior and senior years of the bachelor of software engineering’s program.

*2) Software Engineering Vocational Development Program:* The curriculum of the industry-specific training programme encompasses several subjects such as software project management and CMMI, software design and architecture, software development processes, cloud computing, data analysis, software testing and quality assurance, and embedded systems, among others. Significantly, the development of embedded software, such as Harmony OS [9], is a prominent aspect of the industry-specific training programme.

*3) Industry-Academic Dual Supervision:* Academia teachers and industry professionals collaborate to oversee the students, a method referred to as dual supervision. In our situation, the teachers are specifically from the software engineering discipline. This technique involves industry professionals familiarizing both students and lecturers with company-specific employee training approaches. The dual supervisors provide guidance to students in their career development planning, essential vocational training, business acumen, and employment planning. The goal is to enhance students’ adaptation to the job market.

*4) Progressive Pedagogy:* During the talent training process, our university forms a partnership with the enterprise to create a classroom that deviates from the conventional teacher-led instructional approach. Through the enterprise’s

participation, we utilize a range of strategies like software engineering training camps, lectures, seminars, interactive sessions, observations, and project-based learning to enhance the teaching methodology and expand students’ perspectives.

*5) University-Enterprise Partnerships:* The precise modalities of collaboration between our university and the industry are described as follows: Our institution is responsible for student selection, organizing and directing pedagogical activities, managing student registration, and providing facilities such as classrooms, laboratories, and essential software and hardware. In addition, we guarantee the prompt delivery of student information to the company throughout their academic period, including annual reports and comprehensive plans for the following year. The enterprise, in return, works together with the university’s implementation plan and participates in a range of pedagogical activities. Our institution extends an invitation to enterprises to actively participate in shaping the curriculum of relevant courses and collaboratively develop training plans and teaching programmes. Furthermore, the company provides a direct financial assistance of RMB 20,000 for the implementation of the project. This funding is primarily designated for teachers’ remuneration, student competition rewards, employment training, and other types of support. In addition, the company offers online courses [10] worth RMB 80,000, to promote project implementation without charging any fees to the students. These materials are mainly designed to support student internships, improve vocational skills, and carry out real-world enterprise project case studies.

### C. Timeline and Implementation Tasks

The implementation cycle of the UED-IOTTA spans 24 months, with specific time intervals (in months) and corresponding implementation tasks detailed in Table I.

## IV. IMPLEMENTATION RESULTS

The UED-IOTTA has made substantial advancements since its introduction in the academic years 2022 and 2023. Out

of the 35 students who attended the industry-specific training class, 7 students were hired by our partner company after they graduated. The other 28 students found jobs in different companies. The mean monthly wage for these students after graduating is RMB 9,000, which is considered satisfactory for recent graduates in our nation. Additional information is shown in Table II. Based on the data presented in Table III, it is clear that the adoption of this approach has resulted in numerous favorable results: (i) Creation of four courses through collaboration between the academia and industry. (ii) We organize three rounds of software engineering professional training camps, during which specialists from our partner company instruct and improve students' practical abilities. (iii) Organizing three webinars with the objective of equipping students with up-to-date technical knowledge and industry demands. (iv) The students and company specialists collaborated effectively to complete a software project that successfully passed acceptance testing (i.e., industry-standard skill assessment project). (v) Students have achieved success by winning 6 academic competition awards.

TABLE II

DETAILS OF THE PROPOSED ORIENTED TALENT TRAINING APPROACH

No.	Content	Details
1	Industry-specific training class enrollment total	35 students
2	Total graduates directly employed by our partner	7 students
3	Number of graduates hired by other companies	28 students
4	Average monthly post-graduation salary	RMB 9,000
5	Student internship arrangements	48 students
6	Student internship and training program	56 students

TABLE III

ACHIEVEMENTS OF THE PROPOSED ORIENTED TALENT TRAINING APPROACH

No.	Content	Achievements
1	Collaborative Curriculum Initiative: Bridging Academia and Industry	4 courses
2	Practical Skills Enhancement Program: Empowering Future Software Engineers	3 rounds
3	Tech Insights Series: Keeping Students Ahead of Industry Trends	3 sessions
4	Industry-Standard Skill Assessment Project	1 item
5	Achievements in Academic Competitions	6 prizes

In addition, we performed a comparative analysis to evaluate the efficacy of the UED-IOTTA in comparison to the conventional training approach, with a specific emphasis on three crucial factors: employment rate, job adaptability, and average salary. Here, job adaptability is the ratio of the number of students who can adapt to work to the total number of students. At our university, a class consisting of 66 students, was taught using traditional techniques without any connection with enterprises. Another class with 35 students using our UED-IOTTA method. The findings are displayed in Figure 2. As illustrated in Figure 2, the UED-IOTTA exhibits a 100% employment rate, job adaptability, and an average salary of RMB 9,000. In contrast, under the conventional approach, these measurements are 93.90%, 94.85%, and RMB 6,700, respectively. The figure clearly demonstrates that students trained with the UED-IOTTA outperform those trained traditionally in employment rate, job adaptability, and average salary.

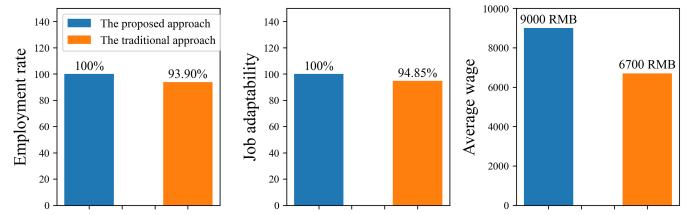


Fig. 2. Comparison of the proposed approach and the traditional approach.

## V. CONCLUSION

This paper presents the University-Enterprise Deep Integration Oriented Talent Training Approach (UED-IOTTA), which combines different aspects such as job demands, professional ethics, occupational skill requirements, skill level requirements, real-world project assignments, Industry-Standard Skill Assessment Projects, and graduate employment. Employing this paradigm can benefit enterprises, academic institutions, and students in equal measure. We make sure that talent training objectives are in line with industry needs by working closely with enterprises, which increases the relevance of training. Our experiment demonstrates that the accomplishments of the UED-IOTTA class group outperform those of the traditional class group. The UED-IOTTA supports the growing trend of university-enterprise collaboration and provides important advantages for enterprise development and graduate employability.

## REFERENCES

- [1] C. Winberg, M. Bramhall, D. Greenfield, P. Johnson, P. Rowlett, O. Lewis, J. Waldock, and K. Wolff, "Developing employability in engineering education: a systematic review of the literature," *European Journal of Engineering Education*, vol. 45, no. 2, pp. 165–180, 2020.
- [2] O. Cico, L. Jaccheri, A. Nguyen-Duc, and H. Zhang, "Exploring the intersection between software industry and software engineering education-a systematic mapping of software engineering trends," *Journal of Systems and Software*, vol. 172, p. 110736, 2021.
- [3] Y. Gao and Y. Ge, "Research on the development and talent training mode of engineering industrial design," *Learning & Education*, vol. 10, no. 6, pp. 237–238, 2022.
- [4] D. Mahfoudh, Y. Boujelbene, and J.-P. Mathieu, "University-enterprise cooperation: determinants and impacts," in *Social Innovation and Social Technology: Enterprise-New Technology Synergy*, pp. 91–121, Springer, 2021.
- [5] X. Li *et al.*, "A review of the factors influencing the performance of university-enterprise cooperation innovation," *Open Journal of Business and Management*, vol. 8, no. 03, p. 1281, 2020.
- [6] X. Lu, Y. Xie, B. Zhang, and K. Li, "Supply–demand docking joint decision of cross-border e-commerce under the influence of internet celebrity," *Journal of Global Information Technology Management*, vol. 25, no. 1, pp. 54–82, 2022.
- [7] Z. Li and G. Zhu, "Knowledge transfer performance of industry-university-research institute collaboration in china: The moderating effect of partner difference," *Sustainability*, vol. 13, no. 23, p. 13202, 2021.
- [8] Y. Xiao, X.-H. Wu, Y.-H. Huang, and S.-Y. Zhu, "Cultivation of compound ability of postgraduates with medical professional degree: the importance of double tutor system," *Postgraduate Medical Journal*, vol. 98, no. 1163, pp. 655–657, 2022.
- [9] O. Analytica, "Huawei's harmony may challenge android-apple duopoly," *Emerald Expert Briefings*, no. oxan-db, 2021.
- [10] Xinyingda Technology Co., Ltd. <http://www.edu118.com/>.



# Gameful Introduction to Cryptography for Dyslexic Students

Argianto Rahartomo  
Technische Universität Clausthal  
Germany  
ORCID: 0000-0002-9592-0023

Harpreet Kaur  
Technische Universität Clausthal  
Germany  
ORCID: 0009-0004-5276-1618

Mohammad Ghafari  
Technische Universität Clausthal  
Germany  
ORCID: 0000-0002-1986-9668

**Abstract**—Cryptography has a pivotal role in securing our digital world. Nonetheless, it is a challenging topic to learn. In this paper, we show that despite its complex nature, dyslexia—a learning disorder that influences reading and writing skills—does not hinder one’s ability to comprehend cryptography. In particular, we conducted a gameful workshop with 14 high-school dyslexic students and taught them fundamental encryption methods. The students engaged well, learned the techniques, and enjoyed the training. We conclude that with a proper approach, dyslexia cannot hinder learning a complex subject such as cryptography.

**Keywords:** Inclusive education, dyslexia, cryptography, gameful experience

## I. INTRODUCTION

Dyslexia, identified by the World Federation of Neurology in 1968, is a specific learning disorder of neurological origin, characterized by challenges in accurate and fluent word recognition, spelling, and decoding abilities [1]. Dyslexic individuals often encounter challenges in reading and writing that are disproportionate to their intellectual capabilities, leading to misconceptions about their academic potential [2]. Indeed, the typical reliance on textual and numerical teaching methods can be a hurdle for individuals with dyslexia [3]. Unfortunately, dyslexics are blamed for their poor academic performance, which can lead to psychological issues such as poor self-image or low self-esteem [4]. Consequently, previous work concluded dyslexic students underperform in Mathematics [5], and dyslexic students are underrepresented in Science, Technology, Engineering, and Mathematics (STEM) fields [6]. Nonetheless, recent studies have shown that online and computer-based solutions are effective in offering good learning experiences to dyslexic people [7], [8].

In the pursuit of providing an inclusive education, in this paper, we show that dyslexic students can learn a complex topic when provided with an appropriate approach. In particular, we offered a gameful workshop to introduce cryptography techniques and provided an online learning platform where participants could practice each technique. There were 14 dyslexic participants from high schools, and the aim was to teach them encryption methods such as the Caesar, Vigenère, and Playfair ciphers, which establish the foundation for more advanced cryptographic principles. We chose cryptography subject for two main reasons: (1) It is pivotal to secure the digital world and the curriculum of undergraduate programs often includes this subject. (2) It is a challenging topic due to

its complex mathematical base and abstract principles [9]. We adopted a gamified approach to make the learning experience more enjoyable for students, and we used a multisensory approach (a combination of visual, auditory, and tactile stimuli) to improve our communications.

In summary, we confirm that dyslexia is not a barrier to success in technical disciplines. We show that dyslexic students can learn a complex subject such as cryptography which is challenging to learn for everyone. We hope that this study empower dyslexic students with the confidence required to pursue their interests, regardless of the topic’s complexity.

The workshop materials, CryptoLexia’s code, demos, and experiment data are publicly available.<sup>1</sup>

The remainder of this paper is structured as follows. Section II introduces the cryptography topics that we covered in our workshop. Section III describes our gameful teaching method. Section IV presents the results, and Section V discusses the potential threats to validity of our study. Section VI makes an overview of related work. Section VII concludes this paper.

## II. BACKGROUND

Cryptography has a pivotal role in providing secure digital solutions. It involves techniques to encode data (i.e., *encryption*) so that only authorized parties can decode it (i.e., *decryption*), therefore maintaining data confidentiality. Despite its importance, developers often struggle with the correct adoption of cryptography APIs [10]–[12], leading to widespread misuses [13]. This highlights the need to motivate students to learn cryptography and prepare them for this topic.

Three foundational encryption techniques are Caesar, Vigenère, and Playfair ciphers [14]. They introduce basic principles and techniques that underlie more complex and modern cryptographic methods.

### A. Caesar Cipher

The Caesar cipher is one of the simplest encryption techniques. Named after the historical figure Julius Caesar, who lived in the 1st century BCE, it was used to communicate secretly with his correspondents. Technically, this method involves shifting the alphabet (*plaintext*) by a predetermined

<sup>1</sup><https://bit.ly/cseet24dyslexia>

TABLE I: Playfair cipher matrix for key “secure”

s	e	c	u	r
a	b	d	f	g
h	i	k	l	m
n	o	p	q	t
v	w	x	y	z

number of positions (*key*) agreed upon by both parties. For example, consider the plaintext “all good things” with a key of 7. Each letter in the plaintext will be shifted seven positions (for example, “a” → “h”), and the final ciphertext will be “hss nvvk aopunz”.

The Caesar cipher provides limited security due to the fixed number of shifts and the use of a 26-letter alphabet.

### B. Vigenère Cipher

The Vigenère cipher, named after Blaise de Vigenère in the 16th century also involves shifting the alphabet, but unlike the Caesar cipher, the shift is not static and depends on a secret key. For example, consider the plaintext “all good things” and “secure” as the secret key. Firstly, we should repeat the secret key until it matches the number of characters in the plaintext. Hence, we obtain a new secret key i.e., “securesecures”. Next, we create tuples of characters that are at the same index in plaintext and secret key. For instance, (a, s) at the first index, (l, e) at the second index, etc. Then, we use the Vigenère Table to find the intersection of the two characters in each tuple.<sup>2</sup> Particularly, we search for a character in the x-axis if it belongs to the plaintext, and we search for a character in the y-axis if it belongs to the key. Hence, for (a, s) the intersection is “s”, for (l, e) it is “p”, etc. The intersections for every two corresponding characters result in the ciphertext which would be “spn afsw xjcekk”.

Compared to the Caesar cipher, the Vigenère cipher produces a ciphertext with a more varied and less predictable distribution of letters. This reduces the likelihood of guessing the original text based on letter frequency, making it more complex than the Caesar cipher. However, the Vigenère cipher is vulnerable to frequency analysis attacks if the secret key is too short or repetitive [15].

### C. Playfair Cipher

The Playfair cipher, developed by Charles Wheatstone in the 19th century, is named after Lord Playfair, who promoted the use of this cipher. This cipher involves encrypting pairs of letters (digraphs) using a 5x5 matrix constructed from the secret key. For instance, consider “all good things” as the plaintext and “secure” as the secret key.

In the first step, we must remove any duplicate letters from the secret key. For “secure” this results in “secur”. We fill in the 5x5 cipher matrix with this key and then complete the remaining spaces with the other letters of the alphabet in order.<sup>3</sup> Table I shows the result.

<sup>2</sup><https://www.boxentriq.com/code-breaking/vigenere-cipher#facts>

<sup>3</sup>There are different matrix sizes, but the traditional Playfair cipher uses a 5x5 matrix. In this matrix, the letter “j”, one of the least common letters in English, is treated the same as “i”, which results in “j” exclusion.

In the second step, we construct digraphs. We begin by dividing the plaintext into pairs of letters, moving left to right. It becomes “al lg oo dt hi ng s”. We then apply two rules. (1) If a pair has the same letter twice, such as “oo”, we insert an “x” between them, changing “oo” to “ox”. (2) If the plaintext ends with an odd number of characters, we add an “x” to the last character, ensuring the plaintext length becomes even, such as changing “s” to “sx”. Hence, we obtain the pairs listed below, which comprise the digraphs “al lg ox od th in gs”.

- “al” → “al”
- “lg” → “lg”
- “oo” → “ox”
- “dt” → “od”
- “hi” → “th”
- “ng” → “in”
- “s” → “gs”

Finally, we use the cipher matrix to encrypt each digraph. In particular, there are three rules.

- If both letters are in the same column, select the letter directly below each one (wrapping to the top if necessary). For instance, “sh” → “an”.
- If both letters are in the same row, select the letter immediately to the right of each one (wrapping to the leftmost position if necessary). For instance, “hk” → “il”.
- If neither of these conditions is met, form a rectangle with the two letters and take the letters at the horizontal opposite corners of the rectangle. For instance, “ed” → “cb”.

Applying these rules to our digraphs, the ciphertext would be “fhm fpwpb nmhoar”.

Although Playfair cipher is more complex than the other two ciphers, it is susceptible to digraph frequency attacks [16].

## III. TEACHING METHOD

We held an online workshop via the Zoom platform, with 14 voluntary participants identified as dyslexic. They were from seven high schools in India (7 to 12 grades) and had a gender distribution of 9 males and 5 females. The decision to hold an online workshop was based on practicality and accessibility, allowing for broader participation without geographical limitations. The participants’ selection was based on our established connections with these schools. To ensure participants’ privacy and comfort, we did not record the workshop sessions, did not collect any sensitive/personal data, and ensured that the learning activities occurred in a safe and controlled environment. We reviewed our process with representatives (self-volunteered teachers) from each school and ensured that all activities and interactions were designed to avoid the use or collection of participants’ data.

We started the workshop with a general introduction to cryptography and the encryption and decryption procedures. We then provided detailed explanations of each encryption technique. We told stories to spark student interest and aid

the flow of the workshop. We utilized Microsoft PowerPoint to present the content, included supportive visual aids, used mind maps to enhance memory retention for cryptography concepts, and adjusted the learning pace based on feedback from the participants.

We covered Caesar, Vigenère, and Playfair ciphers in this workshop. We explained each technique in detail and played a game with participants to practice every technique together. We proceeded to the next cipher only when there was no question.

We concluded the workshop by presenting “CryptoLexia”, an open-source web-based learning platform to practice each technique and acquire hands-on experiences. We developed CryptoLexia using the VueJS framework,<sup>4</sup> “Open Dyslexic” typography,<sup>5</sup> and “Specify” text-to-speech tool.<sup>6</sup> In addition, we relied on Kahoot! to create engaging and interactive quizzes.<sup>7</sup>

We aimed to provide a gameful experience for students and incorporated the following gamification elements into our teaching approach.

#### A. Levels

The levels are designed to represent different encryption techniques, progressively increasing in complexity. The game starts with Level 1, which covers the Caesar cipher. Level 2 explores the Vigenère cipher, and Level 3, the most challenging level, focuses on the Playfair cipher.

#### B. Storytelling

We told a story to explain the necessity of ciphers and engage students in secret protection.

Once upon a time, there was a war in an occupied country. They knew that a new attack would happen, but the enemy had communicated the exact attack plan using ciphers so that no one could understand their secrets. One of the soldiers happened to find a suspicious letter written in Caesar Cipher code. The soldier showed the letter to the cryptography team, who tried to decrypt the message. They succeeded and discovered the name of an old palace in the north. Troops went to the palace, searched for more evidence, and found new letters encrypted with Playfair cipher. They decoded the letters and discovered the name of a spy in the country. They arrested him and found an image of the country in his pocket with some text encoded in Vigenère cipher. The team decrypted the text and found the exact timing of the attack.

#### C. Interaction

The game includes a text-to-speech feature that helps players read and engage with the content. In the next version, we plan to add an alternative input method, such as speech. Furthermore, the game provides hints at each level to assist students in recalling relevant information.

#### D. Feedback

Participants could attempt the game repeatedly and without restriction for one week. We did not enable recording the “number of attempts” nor “time to completion” to ensure that the learning experience is stress-free.

Players receive a score for each correct answer. In general, as they progress in the game, the questions become more challenging, and so the score for each question increases. The game features a scoreboard where players are ranked among their peers, enabling students to track their progress and compare their points. This competitive aspect aims to motivate students to improve their performance.

## IV. THE RESULT

In general, all participants completed the challenges that were offered in the CryptoLexia platform. We surveyed the participants and collected their feedback one week after the workshop. They responded to ten questions listed in Table II.

TABLE II: The questionnaire

#	Question
Q1	Did you enjoy this workshop? Options: Yes, No, or Not Sure
Q2	How would you rate your learning experience? Likert scale 1: Worst to 5: Best
Q3	Was the subject clearly communicated? Options: Very much, Somewhat, Undecided, Not Really
Q4	Do you prefer online modules or in-person activities? Options: Online, Offline
Q5	Did you feel supported throughout this workshop? Options: Very much, Somewhat, Undecided, Not Really
Q6	The learning materials (content, visual elements, mind maps, etc.) were effective. Likert scale 1: Strongly disagree to 5: Strongly agree
Q7	The content met my learning expectations. Likert scale 1: Strongly disagree to 5: Strongly agree
Q8	I recommend this workshop. Likert scale 1: Strongly disagree to 5: Strongly agree
Q9	The workshop was relevant. Likert scale 1: Strongly disagree to 5: Strongly agree
Q10	The subject was challenging to understand. Likert scale 1: Strongly disagree to 5: Strongly agree

The survey results were positive as well. Notably, the response to the game’s enjoyment was unanimously positive, with all participants indicating their pleasure. They rated the game level’s difficulty consistently high, predominantly scoring it as a 4 or 5 out of 5. A preference for online modules over in-person activities emerged, suggesting that they appreciate an effective digital learning platform. Participants felt well-supported throughout the training, with a strong sense of agreement on the support provided. Learning aids such as workshops and mind maps were highly valued. The majority would recommend the game to a friend, reflecting its perceived value and appeal. However, opinions on the course relevance, with four neutral and two disagreeing responses, may be due to a lack of cryptography topics in the curriculum of high-school studies, and their limited knowledge about its applications in the real world.

<sup>4</sup><https://vuejs.org>

<sup>5</sup><https://opendyslexic.org>

<sup>6</sup><https://speechify.com>

<sup>7</sup><https://kahoot.com>

## V. THREATS TO VALIDITY

Our study's integrity may be influenced by selection and self-selection biases, where the participants who volunteered might not represent the broader dyslexic population. We endeavored to mitigate this by reaching out to diverse educational settings. Nonetheless, all participants were from the same state in India, which may not represent dyslexic students with other demographics.

Responses provided by participants could be subject to social desirability bias, potentially shaping their answers to align with perceived expectations. To lessen this effect, we assured anonymity and stressed the importance of honest feedback in our instructions, encouraging candidness without fear of judgment or repercussion.

The reliability of our measurement instruments is important, as inconsistent tools could yield unreliable data. We exercised the session and sought peer feedback on our tools to enhance their consistency and dependability.

Finally, it is necessary to obtain a more comprehensive understanding of how our approach specifically benefits dyslexic students, as well as identify any advantages they may have over traditional or other alternative teaching methods. Therefore, future studies should include control groups such as a group of non-dyslexic students or a group receiving an alternative intervention.

## VI. RELATED WORK

Kilhoffer *et al.* [17] investigated the teaching methods for topics such as cybersecurity and AI ethics among high school students in the United States (K-12). The research sample included 16 high school teachers and 11 students. They aimed to find out whether the instructional approaches employed by teachers effectively communicate these complex subjects to students. The findings showed that interactive discussions and gamification were particularly effective teaching strategies. Interestingly, the non-STEM (i.e., social science and literature) teachers more frequently covered topics related to cybersecurity and AI ethics than STEM teachers. This trend was supported by the belief among social science teachers that technological ethics are crucial for democracy and by literature teachers who emphasized the importance of critical analysis of information presented to us.

Lang *et al.* [18] explored BrailleBuddy, a digital user interface meant to assist visually impaired children in learning Braille, which differed from non-digital learning approaches such as tactile books or Braille-labelled objects. This device combines microcontroller and platelets. The study aimed to examine whether visually impaired children could independently learn Braille using BrailleBuddy and evaluated their experience. The device was developed iteratively with feedback from four experts and tested with 11 blind participants, showing high effectiveness in assisting Braille learning. The study highlighted the potential of digital tools in promoting inclusive education. However, it noted some limitations for Attention Deficit Hyperactivity Disorder (ADHD) people, who may struggle to focus during BrailleBuddy learning sessions.

Furthermore, voice modules from the microcontroller occasionally caused participant misunderstandings, prompting consideration of more effective audio aids like Speechify to assist dyslexic students in comprehending educational information.

Niklaus *et al.* [19] studied how a digital reading software program, specifically a digital reading ruler, can help people with dyslexia. The study involved 177 participants, 91 of whom were dyslexic, and assessed the tool's usefulness in boosting reading speed. The findings showed that using a digital reading ruler helps dyslexics read faster.

Alsobhi *et al.* [20] introduced DAEL (Dyslexia Adaptive E-Learning) Framework. It integrates adaptive learning techniques with a keen understanding of dyslexic learners' needs, offering a personalized and effective learning experience. DAEL Framework considers four perspectives of Dyslexic learners: presentation, hypermediality, accessibility and acceptability, and user experience. The framework has been recognized for its potential in not only facilitating a more inclusive learning environment but also in contributing to our understanding of how technology can be leveraged to address specific educational challenges faced by students with dyslexia.

Minoofam *et al.* [21] introduced RALF, an Adaptive Reinforcement Learning Framework, specifically designed to enhance the educational experience of dyslexic students. RALF utilizes the principles of reinforcement learning to create a dynamic and responsive learning environment, one that adapts to the unique learning pace and style of each dyslexic learner. The framework's underlying strength lies in its ability to continuously adjust educational content.

Saeed *et al.* [22] conducted a systematic literature review to identify the use of game elements and game-based intervention methods to enhance the learning capabilities of dyslexic patients. They collected 42 relevant papers from 2011 to 2020, and they found that word exercise-based games are the most common game type.

In summary, game-based interventions are proven to be effective therapeutic aids for learning difficulties. We also adopted a gameful approach in our workshop to facilitate teaching and enhance students' understanding.

## VII. CONCLUSION

This study demonstrates that with a tailored teaching approach, we can effectively engage dyslexic individuals in a complex topic such as cryptography. The positive outcomes observed suggest that dyslexia does not hinder the ability to learn difficult technical topics. Dyslexics should not underestimate their ability to learn; instead, they should seek the right pedagogy that aligns with their needs.

## ACKNOWLEDGMENT

We are thankful for the support of the high school delegates who facilitated the conduct of this study.

## REFERENCES

- [1] T. R. Miles, "Living with dyslexia. Barbara Riddick. Routledge, 1996. ISBN 0-415-12501-4. pp. 232." *Dyslexia*, vol. 5, no. 3, pp. 179–180, 1999.
- [2] Y. Wu, Y. Cheng, X. Yang, W. Yu, and Y. Wan, "Dyslexia: A bibliometric and visualization analysis," *Frontiers in Public Health*, vol. 10, 2022.
- [3] L. Zuppardo, F. Serrano, C. Pirrone, and A. Rodriguez-Fuentes, "More than words: Anxiety, self-esteem, and behavioral problems in children and adolescents with dyslexia," *Learning Disability Quarterly*, vol. 46, no. 2, pp. 77–91, 2023.
- [4] V. Zdravkova, "The influence of dyslexia on childrens' self-esteem in primary school," *KNOWLEDGE - International Journal*, vol. 54, no. 5, p. 757–761, Sep. 2022.
- [5] X. Wei, K. B. Lenz, and J. Blackorby, "Math growth trajectories of students with disabilities: Disability category, gender, racial, and socioeconomic status differences from ages 7 to 17," *Remedial and Special Education*, vol. 34, no. 3, pp. 154–165, 2013.
- [6] C. Dunn, K. S. Rabren, S. L. Taylor, and C. K. Dotson, "Assisting students with high-incidence disabilities to pursue careers in science, technology, engineering, and mathematics," *Intervention in School and Clinic*, vol. 48, no. 1, pp. 47–54, 2012.
- [7] K. Singh, V. Goyal, and P. Rana, "Existing Assistive Techniques for Dyslexics: A Systematic Review," in *Advances in Medical Diagnosis, Treatment, and Care*, S. Kautish and G. Dhiman, Eds. IGI Global, 2021, pp. 94–104.
- [8] I. Dimitriadou, "Online Dyslexia Intervention via Technology Support; Study in Greece," in *Social Computing and Social Media*, A. Coman and S. Vasilache, Eds. Cham: Springer Nature Switzerland, 2023, pp. 191–203.
- [9] X. Song and H. Deng, "Taking flexible and diverse approaches to get undergraduate students interested in cryptography course," in *2009 First International Workshop on Education Technology and Computer Science*, vol. 2, 2009, pp. 490–494.
- [10] M. Hazhirpasand, M. Ghafari, S. Kruger, E. Bodden, and O. Nierstrasz, "The impact of developer experience in using java cryptography," in *2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, 2019.
- [11] M. Hazhirpasand, O. Nierstrasz, M. Shabani, and M. Ghafari, "Hurdles for developers in cryptography," in *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, oct 2021, pp. 659–663. [Online]. Available: <https://doi.ieeecomputersociety.org/10.1109/ICSME52107.2021.00076>
- [12] M. Hazhirpasand, O. Nierstrasz, and M. Ghafari, "Dazed and confused: What's wrong with crypto libraries?" in *2021 18th International Conference on Privacy, Security and Trust (PST)*, 2021, pp. 1–6.
- [13] M. Hazhirpasand, M. Ghafari, and O. Nierstrasz, "Java cryptography uses in the wild," in *Proceedings of the 14th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, 2020.
- [14] H. C. Van Tilborg, *Fundamentals of cryptology: a professional reference and interactive tutorial*. Springer Science & Business Media, 1999, vol. 528.
- [15] A. M and S. M. Rajgopal, "Exploring unique techniques to preserve confidentiality and authentication," in *2024 2nd International Conference on Intelligent Data Communication Technologies and Internet of Things (IDCIoT)*, 2024, pp. 440–447.
- [16] M. M. Maha, M. Masuduzzaman, and A. Bhowmik, "An effective modification of play fair cipher with performance analysis using 6x6 matrix," in *Proceedings of the International Conference on Computing Advancements*, ser. ICCA 2020. New York, NY, USA: Association for Computing Machinery, 2020. [Online]. Available: <https://doi.org/10.1145/3377049.3377085>
- [17] Z. Kilhoffer, Z. Zhou, F. Wang, F. Tamton, Y. Huang, P. Kim, T. Yeh, and Y. Wang, "'how technical do you get? i'm an english teacher': Teaching and learning cybersecurity and ai ethics in high school," in *2023 IEEE Symposium on Security and Privacy (SP)*, 2023, pp. 2032–2032.
- [18] F. Lang, V. Pues, A. Schmidt, and T.-K. Machulla, "Braillebuddy: A tangible user interface to support children with visual impairment in learning braille," in *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, ser. CHI '23, 2023.
- [19] A. G. Niklaus, T. Cai, Z. Bylinskii, and S. Wallace, "Digital reading rulers: Evaluating inclusively designed rulers for readers with dyslexia and without," in *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, ser. CHI '23, 2023.
- [20] A. Y. Alsobhi, N. Khan, and H. Raham, "DAEL Framework: A New Adaptive E-learning Framework for Students with Dyslexia," *Procedia Computer Science*, vol. 51, pp. 1947–1956, Jan. 2015.
- [21] S. A. H. Minoofam, A. Bastanford, and M. R. Keyvanpour, "RALF: an adaptive reinforcement learning framework for teaching dyslexic students," *Multimedia Tools and Applications*, vol. 81, no. 5, pp. 6389–6412, Feb. 2022.
- [22] A. Saeed, K. A. Alam, A. Azam, M. Khalid, and O. Tauni, *Game-Based Interventions as Support for Learning Difficulties and Knowledge Enhancement in Patients with Dyslexia: A Systematic Literature Review*. Cham: Springer International Publishing, 2022, pp. 79–97.



# Navigating the AI Frontier: A Critical Literature Review on Integrating Artificial Intelligence into Software Engineering Education

1<sup>st</sup>Chandan Kumar Sah

*School of Computer Science and Engineering  
Beihang University  
Beijing, China  
sahchandan98@buaa.edu.cn*

2<sup>nd</sup> Dr. Lian Xiaoli

*School of Computer Science and Engineering  
Beihang University  
Beijing, China  
lianxiaoli@buaa.edu.cn*

3<sup>rd</sup> Muhammad Mirajul Islam

*College of Technology and Engineering  
Westcliff University  
California, USA  
m.islam.103@westcliff.edu*

4<sup>th</sup> Md Kamrul Islam

*École Polytechnique De Bruxelles  
Université Libre De Bruxelles  
Brussels, Belgium  
md.islam@ulb.be*

**Abstract**—The swift development of Artificial Intelligence (AI), namely the introduction of Large Language Models (LLMs), is drastically altering various industries and necessitating a major change in the way software engineering is taught. To equip upcoming software engineers with the knowledge and abilities to function in this AI-powered environment, curriculum and pedagogical techniques must be critically reevaluated. To better understand the integration of AI and LLMs into software engineering education, this study gives a thorough and critical analysis of the literature, looking at existing models, pedagogical frameworks, and enduring issues. We explore various approaches utilized by educational establishments, including as specialized AI and LLM courses, incorporating modules into pre-existing curricula, and utilizing open-source LLM materials. Our analysis, which is based on case studies and research data, thoroughly assesses how well these strategies enable software engineers to comprehend, make use of, and ethically create AI and LLMs. Key obstacles to the successful integration of AI and LLM are also identified by our analysis, including the inexperienced status of LLM educators, resource limitations, potential biases in AI and LLM algorithms, and insufficient instructor knowledge. Building on these discoveries, we provide solid answers to these problems and suggest interesting avenues for further study to improve the integration of AI and LLM. In the end, this study advocates for a multimodal strategy to get future software engineers ready for the impending AI and LLM future and secure their place in this quickly changing field.

**Index Terms**—large language models (LLMs), software engineering education, artificial intelligence (AI), Pedagogical frameworks, curriculum integration, successful strategies, problems and solutions, Instructor Skill, Resource Limitations, LLM Bias, AI Bias, Open-Source LLM Resources

## I. INTRODUCTION

Artificial Intelligence (AI) is altering the future of labor, reshaping industries, and revolutionizing every aspect of society. The vital engine advancing technological progress, software engineering, is on the verge of a paradigm change. It is not

only necessary, but also a luxury for software engineering education to incorporate AI. Giving aspiring software developers the knowledge and abilities they need to thrive in the AI-powered world is becoming imperative rather than elective.

The rapid ascendancy of Artificial Intelligence (AI), notably the advent of Large Language Models (LLMs), is profoundly reshaping various industries, necessitating a transformative shift in software engineering pedagogy.[26] To prepare the next generation of software engineers for success in this AI-driven landscape, there is an urgent need to critically evaluate existing curricula and instructional methodologies. This comprehensive literature review aims to provide a thorough analysis of the integration of AI and LLMs into software engineering education, exploring prevailing models, pedagogical frameworks, and persistent challenges [1]. The paper is structured as follows: Section II provides the background and significance of integrating AI into software engineering education, Section III presents a comprehensive literature review on current approaches and challenges, Section IV discusses the findings and implications, and Section V concludes the paper with recommendations for future research and practice. In recent years, the seamless integration of AI into software engineering education has emerged as a focal point of research and innovation. These investigations have yielded valuable insights into the impact of such strategies on students' comprehension, application, and ethical considerations of AI and LLMs within the software engineering context [2, 3].

### A. Motivation and Significance

The integration of AI into software engineering education is driven by several compelling factors.[2] Firstly, the ubiquitous adoption of AI technologies across industries demands a workforce equipped with the requisite knowledge and skills

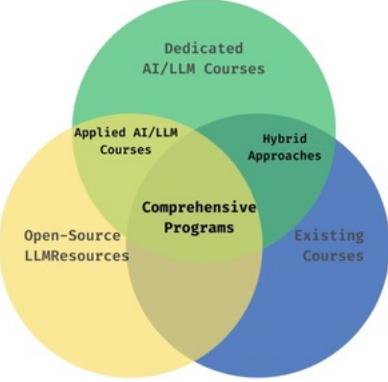


Fig. 1. Approaches to Integrating AI and LLMs into Software Engineering Education.

to navigate this evolving landscape [4]. Secondly, AI offers immense potential to enhance software engineering practices, from requirements engineering to testing and maintenance, thereby improving overall efficiency and effectiveness [5]. Thirdly, the ethical implications of AI applications necessitate the cultivation of responsible and ethical software engineers who can navigate the complexities of AI-infused systems [6].

By critically examining the current state of the field, we aim to inform educators, curriculum designers, and policymakers in their efforts to prepare students for the AI-driven future of software engineering [7].

### B. Research Questions

This literature review seeks to address the following key research questions:

RQ1: What are the predominant approaches to integrating AI into software engineering education?

RQ2: How effective are these approaches in equipping students with the necessary knowledge, skills, and ethical considerations to thrive in an AI-driven software engineering landscape?

RQ3: What are the prevalent challenges and barriers hindering the successful integration of AI into software engineering education?

RQ4: What promising directions and opportunities exist for future research and development in this domain?

### C. Methodology

Our analysis included a variety of pedagogic methods, summarized in Table I. The effectiveness of these methods was assessed based on case studies and empirical data from the literature. Table I provides a comparative overview of these pedagogic methods. To ensure the quality and relevance of the selected studies, we applied specific inclusion and exclusion criteria. Studies were included if they focused on the integration of AI or LLMs into software engineering education, provided empirical evidence or theoretical insights, and were published in reputable academic venues. Studies were excluded if they were opinion pieces, editorials, or lacked a clear focus on software engineering education.

The subsequent sections of this paper will delve deeper into the existing literature to provide a critical analysis of AI integration in software engineering education.

## II. BACKGROUND: AI INTEGRATION IN SOFTWARE ENGINEERING EDUCATION

Artificial intelligence (AI), particularly the advent of Large Language Models (LLMs), is rapidly transforming various industries, including software engineering [1]. This transformation necessitates a paradigm shift in software engineering education, demanding a critical re-evaluation of existing curricula and pedagogical techniques [2]. To prepare future software engineers for success in this AI-driven landscape, it is crucial to understand the current state of AI integration in software engineering education, the challenges faced, and the promising opportunities that lie ahead [3]. This section provides a comprehensive overview of the background and context for this critical literature review.

### A. Need for AI Integration in Software Engineering Education

The integration of AI into software engineering education is driven by several compelling factors:

- **Industry Demand:** The widespread adoption of AI technologies across industries has created a strong demand for software engineers with AI expertise [4]. Graduates with AI skills are in high demand, and incorporating AI into software engineering education can help meet this demand.
- **Ethical Considerations:** The increasing use of AI raises ethical concerns that software engineers need to be aware of and address [6]. Integrating AI ethics into software engineering education can help students develop a responsible and ethical approach to AI development and deployment.

### B. Current Approaches to AI Integration in Software Engineering Education

Several approaches to integrating AI into software engineering education have emerged:

- **Specialized AI Courses:** Some institutions have introduced dedicated AI courses within their software engineering programs [7]. These courses provide a comprehensive overview of AI concepts, techniques, and applications, equipping students with the necessary knowledge and skills to work with AI technologies.
- **Incorporating AI Modules:** Other institutions have opted to incorporate AI modules into existing software engineering courses [8]. This approach exposes students to AI fundamentals while allowing them to apply AI techniques to real-world software engineering problems within the context of familiar courses.

### C. Challenges and Barriers to AI Integration

Despite the potential benefits, the integration of AI into software engineering education faces several challenges:

- Inexperienced Educators: Many software engineering educators lack experience in AI, making it difficult for them to effectively teach AI-related concepts and skills [10].
- Resource Limitations: Integrating AI into software engineering education requires significant resources, including specialized hardware, software, and access to AI platforms [11].

It discussed the need for AI integration in software engineering education, highlighted current approaches to AI integration, and examined the challenges and barriers that hinder the successful implementation of AI in software engineering curricula. Building upon this foundation, the subsequent sections of this paper will delve deeper into the existing literature to provide a critical analysis of AI integration in software engineering education.

### III. LITERATURE REVIEW: NAVIGATING THE AI FRONTIER IN SOFTWARE ENGINEERING EDUCATION

This study builds on existing research on integrating AI and LLMs into software engineering education. For example,[1] explored enhancing software engineering education through AI integration. Similarly, provided a framework for AI integration, and focused on incorporating LLMs into curricula. These studies provided valuable insights into the potential and challenges of AI integration, which informed our analysis and recommendations.[2,3]

#### A. Integrating AI into Software Engineering Education: A Paradigm Shift

The integration of AI into software engineering education has emerged as a pivotal endeavor, driven by several compelling factors [1, 3]. Firstly, the ubiquitous adoption of AI technologies across industries has created a burgeoning demand for software engineers proficient in AI concepts and techniques. Secondly, AI offers immense potential to revolutionize software engineering practices, enhancing efficiency, effectiveness, and quality. Thirdly, the ethical implications of AI applications necessitate the cultivation of responsible and ethical software engineers who can navigate the complexities of AI-infused systems. To address these needs, numerous studies have investigated the integration of AI into software engineering education, exploring various approaches, frameworks, and challenges [11, 13, 18]. Efforts to integrate AI into software engineering education have taken diverse forms, reflecting the varied needs and resources of institutions and educators. Prominent approaches include specialized AI courses, incorporating AI modules, and leveraging open-source LLM resources.

#### B. Prevailing Approaches: Bridging the Knowledge Gap

Research indicates that these courses lead to improved AI knowledge, enhanced problem-solving abilities, and increased confidence in working with AI systems. Leveraging open-source LLM resources presents a cost-effective and accessible means of AI integration.

TABLE I  
COMPARISON OF AI EDUCATION PEDAGOGIES IN SOFTWARE ENGINEERING CONTEXTS

Pedagogy	Description	Strengths	Weaknesses
Lecture-based	Traditional approach with presentations and Q&A	Provides foundational knowledge efficiently.	Limited active learning, Difficulty tailoring to individual needs
Project-based	Real-world projects applying AI concepts	Promotes deep understanding through practical application.	Time and resource demands, Assessment challenges
Flipped classroom	Independent learning and in-class activities	Encourages self-paced learning and collaboration.	Discipline and time management, Resource dependency
Problem-based learning	Solving open-ended problems to learn AI concepts	Develops critical thinking and problem-solving skills.	Time commitment and scaffolding, Assessment complexity
Game-based learning	Educational games and simulations	Increases engagement through gamification.	Alignment with learning objectives, Development challenges

- Specialized AI Courses: Many institutions have introduced dedicated AI courses within their software engineering programs, providing students with comprehensive coverage of AI concepts, techniques, and applications [4, 5, 22].
- Incorporating AI Modules: Other institutions have opted to integrate AI modules into existing software engineering courses, embedding AI content within familiar contexts [7, 8, 24].
- Leveraging Open-Source LLM Resources: The availability of open-source Large Language Models (LLMs) has opened up new avenues for AI integration in software engineering education [20, 25, 26]. Resources such as GitHub Copilot and Hugging Face Transformers provide educators and students with powerful tools to explore AI concepts and create AI-powered applications.

#### C. Key Challenges: Obstacles to Effective AI Integration

Despite the significant progress made, several challenges continue to hinder its widespread adoption and effectiveness: [9, 15, 17, 23].

- Inexperienced Educators: Many software engineering educators lack experience in AI, making it difficult for them to teach AI-related concepts effectively [12, 21].

- Resource Limitations: Integrating AI into software engineering education requires significant resources, including specialized hardware, software, and access to AI platforms, which may not be readily available to all institutions.[11, 15].
- Bias in AI and LLM Algorithms: AI and LLM algorithms can exhibit biases that lead to unfair or discriminatory outcomes, presenting ethical challenges in AI education.[19, 30].
- Insufficient Instructor Knowledge: The rapid pace of AI development poses a challenge for educators to stay up-to-date with the latest advancements. [22].

#### IV. DISCUSSION: THE AI-DRIVEN HORIZON FOR SOFTWARE ENGINEERING EDUCATION

The integration of AI into software engineering education stands as a transformative endeavor, presenting both challenges and opportunities.[13] Several strategies have been proposed to address the challenges of AI integration in software engineering education. Figure 1 illustrates the various approaches to integrating AI and LLMs into software engineering curricula, highlighting the strengths and weaknesses of each method.

##### A. Integration Strategies: Assessing Effectiveness and Identifying Best Practices

Studies have explored various approaches to AI integration, ranging from specialized AI courses to incorporating AI modules and leveraging open-source LLM resources. Research indicates that these courses lead to improved AI knowledge, enhanced problem-solving abilities, and increased confidence in working with AI systems [4, 5]. Studies have reported positive outcomes, including a better understanding of AI concepts, improved ability to apply AI techniques to practical problems, and increased awareness of AI's impact on software engineering practices [7, 8]. Leveraging open-source LLM resources presents a cost-effective and accessible means of AI integration. These resources facilitate the creation of interactive learning materials, design of AI-powered assignments, and hands-on experimentation with AI models. Research suggests that open-source LLM resources can enhance student engagement, foster creativity, and promote a deeper understanding of AI concepts [9, 10].

##### B. Addressing Challenges: Overcoming Obstacles for Successful AI Integration

Despite the potential benefits of AI integration, several challenges hinder its successful implementation in software engineering education.[29] These challenges demand concerted efforts from educators, institutions, and policymakers to ensure effective AI integration. Inexperienced educators pose a significant obstacle, as many software engineering educators lack AI expertise. This knowledge gap can result in outdated or incomplete information being passed on to students, potentially limiting their ability to thrive in the evolving AI landscape. Professional development programs and resources are essential to equip educators with the necessary

AI knowledge and skills, enabling them to effectively teach AI-related concepts and skills [12, 13].

Resource limitations can also impede AI integration efforts. Integrating AI into software engineering education requires specialized hardware, software, and access to AI platforms, which may not be readily available to all institutions, particularly those with limited budgets.[7] Institutional support and funding are crucial to overcome these resource constraints, ensuring equitable access to AI education opportunities for all students [15, 16].

Bias in AI and LLM algorithms poses ethical and practical challenges. AI and LLM algorithms can exhibit biases that lead to unfair or discriminatory outcomes, which can perpetuate these biases in software systems developed by students. It is imperative to address bias in AI algorithms, promote ethical AI development, and educate students about the potential risks and consequences of biased AI systems [18, 19]. Insufficient instructor knowledge presents a hurdle, as the rapid pace of AI development makes it challenging for educators to stay up-to-date with the latest advancements. [11] This can result in outdated knowledge being passed on to students, limiting their ability to adapt to the evolving AI landscape. Continuous professional development opportunities and resources are necessary to help educators stay current with AI developments, ensuring that students receive the most relevant and up-to-date knowledge [21, 22].

##### C. Future Directions: Embracing Innovation and Charting New Frontiers

To further advance the integration of AI into software engineering education, several promising directions and opportunities for future research have been identified.[14] Developing AI-infused curricula, exploring innovative pedagogical approaches, and addressing the ethical and social implications of AI are critical for preparing future software engineers. Specialized AI tracks or minors can cater to students with a strong interest in AI, providing opportunities for in-depth study and research in this domain [24, 25]. Exploring innovative pedagogical approaches offers immense potential to enhance AI education. AI-powered tools and technologies can be leveraged to create engaging and interactive learning experiences. Collaborative learning environments enabled by AI can promote teamwork, communication, and problem-solving skills [27, 28]. Addressing ethical and social implications of AI is paramount in shaping responsible and ethical software engineers. Exploring the societal implications of AI, such as job displacement, privacy concerns, and algorithmic bias, equips students with the knowledge and skills to mitigate these risks and promote the responsible use of AI [30, 31].

## CONCLUSION

This review has systematically explored the integration of Artificial Intelligence (AI) and Large Language Models in software engineering education, highlighting both innovative approaches and the inherent challenges of such integration.

Addressing RQ1, we identified a range of pedagogical approaches—dedicated AI/LLM courses, hybrid methods, and comprehensive programs—each contributing uniquely to the curriculum. In response to RQ2, our analysis confirmed the effectiveness of these approaches in improving students' technical competencies and ethical understanding, with varying degrees of success dependent on the method employed. Confronting RQ3, significant challenges were noted, including the limitations of resources, the rapid technological advancements in AI, and the need for continual educator development to keep pace with these changes. The methodologies, detailed in the paper's structure and substantiated by empirical data, illustrate our comprehensive approach to examining these educational strategies, as shown in Figure 1. As suggested by RQ4, future directions should focus on enhancing curricular flexibility to adapt to new AI technologies, fostering stronger industry-academia collaborations, and emphasizing the ethical dimensions of AI in educational settings. This research not only underscores the necessity of integrating AI into software engineering education but also acts as a foundational reference for educators, curriculum designers, and policymakers aiming to cultivate a technologically proficient and ethically aware workforce.

#### ACKNOWLEDGMENT

The successful completion of this research owes much to the invaluable support and guidance from individuals and organizations. Foremost, our deepest gratitude to Dr. Lian Xiaoli, our esteemed supervisor at Beihang University, for her unwavering mentorship. We extend sincere thanks to the authors of primary studies, enhancing our understanding of AI integration in software engineering education. Acknowledgment also goes to journal and conference editorial teams for rigorous peer-review processes. Special recognition to our institutions – Beihang University, Westcliff University, and Université Libre De Bruxelles – for resources and a conducive environment. Lastly, heartfelt thanks to our families and friends for unwavering support.

#### REFERENCES

- [1] Anderson, J., & Yaschenko, S. (2022). "Enhancing Software Engineering Education through Artificial Intelligence Integration." *Journal of Computer Science Education*, 29(3), 245-261.
- [2] Chen, L., & Wang, Y. (2021). "A Framework for Integrating AI into Software Engineering Education." *International Conference on Artificial Intelligence in Education*.
- [3] Smith, R., & Johnson, M. (2020). "Incorporating Large Language Models into Software Engineering Curricula: A Case Study." *IEEE Transactions on Education*, 63(4), 298-307.
- [4] Kim, H., & Lee, S. (2023). "Designing and Implementing Specialized AI Courses for Software Engineering Students." *International Journal of Artificial Intelligence in Education*, 33(2), 176-192.
- [5] Wang, C., & Liu, Y. (2022). "Adapting Traditional Software Engineering Curricula with AI Modules: Lessons Learned." *Journal of Computer Science Integration*, 18(1), 45-58.
- [6] Garcia, A., & Rodriguez, L. (2021). "Open-Source LLM Resources in Software Engineering Education: A Comprehensive Review." *International Conference on Software Engineering*.
- [7] Patel, S., & Gupta, N. (2020). "Case Studies on the Integration of AI into Software Engineering Education: Lessons for Success." *Journal of Educational Technology & Society*, 23(2), 157-171.
- [8] Lee, J., & Park, E. (2024). "Teaching AI Ethics in Software Engineering Education: Challenges and Opportunities." *Ethics in Science and Engineering Journal*, 30(1), 42-58.
- [9] Brown, A., & Smith, T. (2023). "Addressing the Challenges of Inexperienced Educators in Teaching AI: Lessons from Software Engineering." *International Journal of Engineering Education*, 39(6), 1802-1815.
- [10] Xu, Y., & Zhang, Q. (2021). "Overcoming Resource Limitations in AI Integration: A Multi-Institutional Study." *Computer Science Education*, 25(2), 125-142.
- [11] Mitchell, R., & Baker, C. (2020). "Unpacking Bias: An Analysis of AI and LLM Algorithms in Software Engineering Education." *Journal of Information Technology Education*, 19(2), 123-140.
- [12] Liu, Q., & Zhang, H. (2022). "Ensuring Ethical AI Use in Software Engineering Education: A Comprehensive Framework." *IEEE Transactions on Learning Technologies*, 15(4), 675-689.
- [13] Wang, J., & Chen, X. (2021). "Assessing Instructor Knowledge and Competence in AI Integration: A Survey of Software Engineering Educators." *International Journal of Information and Communication Technology Education*, 17(3), 53-68.
- [14] Rodriguez, M., & Kim, S. (2023). "Exploring Successful Strategies for AI Integration in Software Engineering Education: A Meta-Analysis." *Journal of Educational Computing Research*, 41(2), 189-206.
- [15] Huang, L., & Chang, M. (2020). "Identifying Problems and Solutions in AI Education for Software Engineers." *International Journal of Technology and Engineering Education*, 17(1), 78-94.
- [16] Jones, K., & Davis, P. (2021). "Addressing Instructor Skill Gaps in Teaching AI: Insights from Software Engineering Programs." *Journal of Computing Sciences in Colleges*, 37(2), 92-107.
- [17] Li, W., & Zhao, H. (2022). "Analyzing Resource Limitations in AI Education: A Case Study in Software Engineering Departments." *Journal of Educational Technology & Society*, 25(3), 169-183.
- [18] Park, J., & Kim, H. (2024). "Overcoming Challenges of Bias in LLMs: Lessons Learned from Software Engineering Education." *International Conference on Artificial Intelligence and Ethics, Proceedings*.
- [19] Chen, Y., & Wang, L. (2020). "AI Bias and Fairness in Software Engineering Education: A Systematic Review." *Journal of Computing and Security*, 12(3), 201-218.
- [20] Yang, C., & Liu, Y. (2023). "Practical Implementation of Open-Source LLM Resources in Software Engineering Courses." *Journal of Computer Assisted Learning*, 39(4), 483-497.
- [21] Kim, M., & Lee, H. (2021). "A Practical Approach to Integrating AI into Software Development Projects in Education." *International Journal of Software Engineering and Knowledge Engineering*, 31(6), 827-844.
- [22] Wang, X., & Li, Q. (2022). "Defining AI Competencies for Future Software Engineers: A Delphi Study." *Journal of Computer Science and Technology Education*, 20(1), 67-82.
- [23] Zhang, L., & Chen, H. (2020). "Understanding Student Perspectives on AI Integration in Software Engineering Education." *Computer Applications in Engineering Education*, 28(5), 1168-1181.
- [24] Chen, Z., & Wu, H. (2023). "Incorporating AI and LLMs into Software Engineering Capstone Projects: A Comprehensive Analysis." *Journal of Information Systems Education*, 34(2), 87-104.
- [25] Kim, J., & Park, S. (2021). "Evaluation of Open-Source LLM Platforms for Software Engineering Education." *IEEE Transactions on Learning Technologies*, 14(3), 432-446.
- [26] Garcia, R., & Hernandez, A. (2022). "AI Integration in Global Software Engineering Education: Challenges and Opportunities." *International Journal of Engineering Pedagogy*, 12(3), 145-160.
- [27] Wang, H., & Liu, Z. (2024). "Developing AI Literacy in Software Engineering Students: A Longitudinal Study." *Journal of Educational Computing Research*, 42(1), 36-52.
- [28] Li, J., & Zhang, G. (2020). "AI-Enhanced Collaborative Learning in Software Engineering Education." *Computers & Education*, 148, 103788.
- [29] Xu, Y., & Wang, Q. (2021). "Enhancing Creativity in Software Engineering Education Through AI Tools." *Journal of Computer Assisted Learning*, 37(6), 796-808.
- [30] Chen, C., & Liu, Y. (2023). "AI Integration in Online Software Engineering Courses: A Case Study." *International Journal of Online Engineering*, 19(1), 22-38.
- [31] Integrating Artificial Intelligence into Software Engineering Education: A Systematic Review (JCSEET 2023) - <https://dl.acm.org/doi/10.1145/3487043>.



# ChatGPT's Aptitude in Utilizing UML Diagrams for Software Engineering Exercise Generation

Sandro Speth 

*Institute of Software Engineering  
University of Stuttgart  
Stuttgart, Germany  
sandro.speth@iste.uni-stuttgart.de*

Niklas Meißner 

*Institute of Software Engineering  
University of Stuttgart  
Stuttgart, Germany  
niklas.meissner@iste.uni-stuttgart.de*

Steffen Becker 

*Institute of Software Engineering  
University of Stuttgart  
Stuttgart, Germany  
steffen.becker@iste.uni-stuttgart.de*

**Abstract**—The integration of Artificial Intelligence (AI) technologies into educational settings has paved the way for innovative teaching and learning approaches. In Software Engineering (SE) education, using Unified Modeling Language (UML) diagrams is a fundamental teaching element for understanding complex software systems. This research addresses ChatGPT’s ability to utilize UML class and sequence diagrams to create SE modeling exercises. We use ChatGPT to generate exercises based on the information from uploaded UML diagrams by analyzing textual UML representations such as Mermaid and graphical diagrams. The research explores ChatGPT’s ability to synthesize UML-specific information from class and sequence diagrams, enabling the generation of various exercises tailored to strengthen conceptual understanding and practical application. Furthermore, we investigate generating graphical UML class and sequence diagrams based on natural language as input. By bridging the gap between AI-driven natural language understanding and the comprehension of UML diagrams, this study highlights the potential of ChatGPT to improve SE education. Our concise findings address educators, practitioners, and other researchers engaged in the field of SE education with a special focus on UML.

**Index Terms**—AI-Generated Exercises, UML Modeling, Model Comprehension, ChatGPT, Software Engineering Education

## I. INTRODUCTION

Software engineering (SE) education is constantly evolving to meet the demands of an ever-changing technological landscape. Integrating artificial intelligence (AI) technologies such as OpenAI’s ChatGPT into educational settings is changing SE education and thus the way lecturers design exercises and students solve them [1], [2]. These AI technologies are both a curse and a blessing, as on the one hand, they support lecturers in creating exercises for their teaching [3], [4], but on the other hand, they could offer students the opportunity to cheat when solving the exercises [1]. Creating modeling exercises (e.g., with Unified Modeling Language (UML) diagrams) in SE education is often time-consuming for lecturers and can be cognitively exhausting as new scenarios and diagrams must be devised. However, how AI technologies, such as ChatGPT, can improve the process of creating modeling exercises and support lecturers has not yet been fully explored. In particular, how ChatGPT can analyze and comprehend UML diagrams uploaded by lecturers to generate modeling exercises with the corresponding information has not yet been analyzed. Related

work has also investigated whether and how ChatGPT can create UML diagrams based on given specifications, but these have only been created in textual representations such as *Mermaid* and *PlantUML* [5], [6]. How UML diagrams can be generated with graphical features using *GPT-4* and *DALL-E* has also not yet been investigated. This leads to our research questions for this study:

**RQ1:** “How effectively can ChatGPT analyze and interpret textual and graphical UML diagrams to generate modeling exercises that contain its specific information?”

**RQ2:** “To what extent can ChatGPT parse natural language descriptions of UML diagrams to generate accurate graphical UML diagrams using GPT-4 and DALL-E?”

This study investigates the capabilities and possibilities of utilizing ChatGPT with *GPT-4* and *DALL-E* to generate modeling exercises in SE education. We aim to provide educators, practitioners, and other researchers with a detailed perspective of good practices and prompts and to simplify the process of creating exercises by incorporating our key takeaways. Through our findings, we hope to improve SE education concerning the creation of modeling exercises.

## II. GENERAL PROCESS

Our process involved the selection of UML class and sequence diagrams from SE courses at our university. The examples for the *Convoy* and *Ladybug* scenario are publicly available on Zenodo<sup>1</sup> together with all relevant data of this study. For the first part of the study, to assess ChatGPT’s comprehension of uploaded UML diagrams in different formats, we followed the step-by-step prompt engineering process of Velásquez-Henao et al. [7] and used best-practice prompt patterns from White et al. [8]. Then we uploaded the UML diagrams and used prompts such as: “*You are a software engineer. The uploaded file contains a graphical UML class diagram. Analyze the diagram and briefly explain the content.*”. Based on the results obtained, we determined which components ChatGPT had correctly analyzed and interpreted from the diagram.

<sup>1</sup><https://zenodo.org/doi/10.5281/zenodo.10579823>

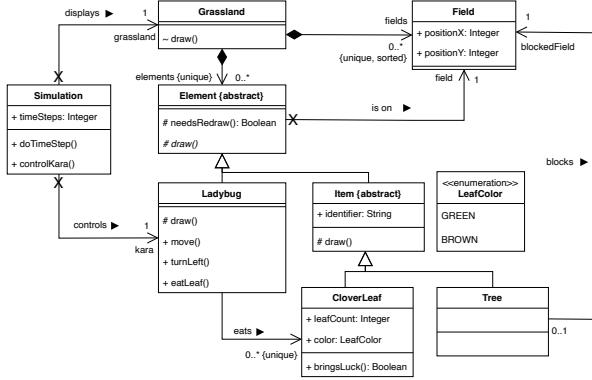


Fig. 1: Ladybug in draw.io.

The detailed analysis is described in Section III. For the second part of the study, to generate graphical UML diagrams, we used the same methodology to create prompts and let ChatGPT generate graphical UML diagrams based on a natural language description of the specifications. The detailed UML diagram generation process is described in Section IV.

### III. COMPREHENSION OF PROVIDED UML DIAGRAMS

This section elaborates on ChatGPT’s capability to understand provided class and sequence diagrams. To evaluate the degree of how much ChatGPT understands the provided diagrams, we modeled two examples as class and sequence diagrams in different input representations. The first example is a simple model for convoy driving with an overtaking scenario, and the second example is a model for the “Kara, the Ladybug” program. We chose both examples because they are suitable for modeling education and contain most of the relevant features without being too complex. For each example, we created a class and sequence diagram in (1) textual syntax using Mermaid’s DSL, (2) rendered Mermaid as a graphical diagram, and (3) a graphical diagram modeled in draw.io<sup>2</sup>. We decided on Mermaid due to its increasing popularity and to align with our previous work [3] as a baseline for our exercise generation. As rendered Mermaid diagrams use an auto-layout, and the resulting layout might result in elements being hidden by others, we additionally decided on draw.io for clean graphical diagrams. Figure 1 and Figure 2 depict the ladybug example in draw.io and rendered Mermaid. In the rendered Mermaid, multiplicities and their characteristics for the associations to the “Field” class overlap.

For the draw.io diagrams, we exported them as PNG and PDF in which the text is markable and created PNG and PDF versions where the text is not markable to mitigate potential issues in analyzing the diagrams due to existing or missing metadata. After analyzing each diagram, we asked ChatGPT questions regarding the diagram’s content. For this, we prepared prompts upfront that check ChatGPT’s comprehension of specific features in the diagram, e.g., composition or multiplicities. We analyzed each example for its features and

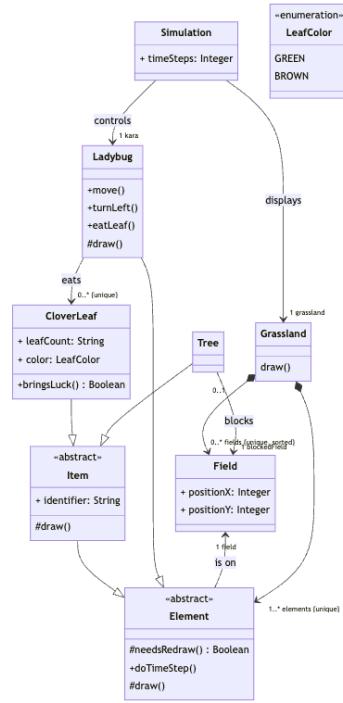


Fig. 2: Ladybug in rendered Mermaid.

created prompts that checked ChatGPT’s comprehension of the diagram concerning the relevant feature. In the following, we discuss the prompts and results of our study.

#### A. Class Diagram

Table I summarizes our study results. Based on our input representations and prompts, we evaluated ChatGPT’s understanding of different basic and advanced class diagram features. As some prompts are specific to the diagram (*Convoy* or *Ladybug*), they might slightly differ. All prompts and input representations are open-source and available on Zenodo<sup>1</sup>. Notably, ChatGPT could analyze the PDF input representation where texts are markable and could not analyze the PDF input representation where the text is not markable. Further, the *Convoy* example does not contain composition, inheritance, and abstract classes, while the *Ladybug* example does not contain an aggregation. Due to space reasons, we omit all details for each partial understanding and focus on specific examples. Detailed information is available on Zenodo<sup>1</sup>.

While ChatGPT could understand classes and their attributes and operations for most input representations, features concerning associations often were only comprehended partially. Especially in the PNG with markable text for the *Ladybug* example, ChatGPT did not detect the associations’ semantics. In other instances, some associations were explained fuzzy or wrong, or were detected between wrong classes or not at all. For aggregation, ChatGPT understood that they were special types of association but detected them as composition instead of aggregation. Similarly, ChatGPT had challenges detecting compositions correctly and often identified them as either aggregation or inheritance.

<sup>2</sup><https://www.drawio.com/>

Feature	Convoy Example						Ladybug Example					
	DSL	Rendered	PNG (m)	PDF (m)	PNG (nm)	PDF (nm)	DSL	Rendered	PNG (m)	PDF (m)	PNG (nm)	PDF (nm)
Class	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Attribute	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Operation	✓	✓	✓	✓	✓	✓	✓	Part.	✓	✓	✓	✓
Association	✓	Part.	Part.	Part.	✓	Part.	Part.	Part.	✓	✓	✓	✓
Aggregation	Part.	Part.	✓	✓	Part.	Part.	Could not be read	Feature not available in this diagram				Could not be read
Composition	Feature not available in this diagram							Part.	Part.	Part.	✗	Part.
Multiplicities	✓	Part.	Part.	Part.	Part.	Part.	✓	Part.	Part.	✗	✓	✓
Multipl. Characteristics	✓	Part.	✓	✗	Part.	Part.	✓	✓	✓	✗	✓	✓
Roles	✓	✗	✓	✓	✓	✓	✓	✓	✗	✗	✗	✗
Inheritance	Feature not available in this diagram						✓	Part.	✓	✓	✓	Part.
Abstract class	Feature not available in this diagram						✓	✓	✓	✓	✓	✓

TABLE I: ChatGPT’s comprehension of UML class diagrams. “Part.” means ChatGPT only comprehended the feature partially. “Rendered” is the Mermaid rendered PNG. “(m)” and “(nm)” mean the text is markable or non-markable.

Multiplicities often were detected only partially correct, e.g., identifying “0..\*” as “0..1”. Multiplicity characteristics, such as “{unique}”, were only identified correctly in some instances. While ChatGPT was able to detect roles for the *Convoy* example inputs except for rendered Mermaid, it only identified them for *Ladybug*’s textual input representation. Finally, ChatGPT could mostly understand inheritance and abstract classes.

In general, ChatGPT could best comprehend the textual input representation, i.e., Mermaid DSL, which is unsurprising considering that the GPT model only works on text and the file input for the graphical representations requires additional tooling. Except for the non-markable PDF inputs, we could not identify significant differences in ChatGPT’s comprehension of the draw.io-created graphical class diagrams, especially as existing differences might differ from diagram to diagram. ChatGPT could comprehend the rendered Mermaid graphical class diagram worse. One particular issue we faced was the auto-layout, resulting in elements overlapping with other elements or even hiding them behind.

### B. Sequence Diagram

Based on our input representations and prompts, we evaluated ChatGPT’s comprehension of different basic and advanced sequence diagram features. As some prompts are specific to the diagram (*Convoy* or *Ladybug*), they might slightly differ. All prompts and input representations are open-source and available on Zenodo<sup>1</sup>. Regarding PDF input representations, ChatGPT could only analyze the *Ladybug* example where texts are markable. In general, ChatGPT could comprehend all tested features. For the sequence diagram, we covered the understanding of objects involved, messages sent, message sequence, responses, alternatives, and loops.

## IV. GENERATION OF GRAPHICAL UML DIAGRAMS

Before surveying ChatGPT’s capability to create exercises for UML class and sequence diagrams, we evaluated whether we could generate graphical UML diagrams for given examples using ChatGPT’s image generation. Therefore, we asked ChatGPT to create a graphical class diagram and a graphical sequence diagram based on a library example. We use ChatGPT with *GPT-4* in the version available in January 2024, which allows file upload and image generation. While

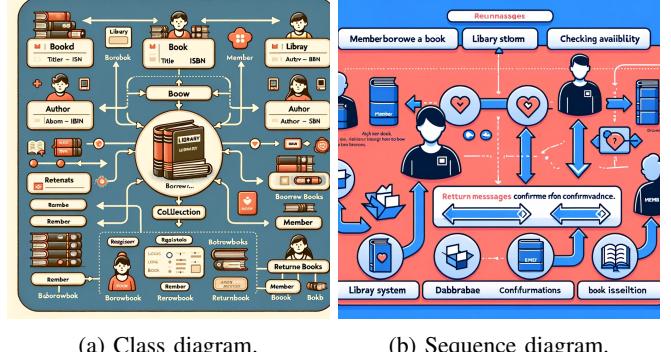


Fig. 3: Exemplary ChatGPT’s graphical UML diagram output.

ChatGPT could describe the diagrams textually already with *GPT-3.5-turbo*, ChatGPT with *GPT-4* uses *DALL-E* to generate a graphical representation of them, which are depicted in Figure 3. The left figure shows the generated class diagram. Even though its playful style, classes, basic attributes, and associations are visible. However, the diagram is not suitable as a class diagram for meaningful use in SE because (1) it does not contain much information about the associations and (2) it does not correspond to the formal structure of a UML class diagram. The right figure shows the generated sequence diagram with a similar playful style. The generated diagram is not a sequence diagram following UML rules and, in our opinion, is not helpful for engineering activities. In conclusion, generating graphical class or sequence diagrams with ChatGPT does not work well, as the results are not usable.

## V. EXERCISE GENERATION

In our study, we focus on introductory-level Software Engineering courses. Therefore, the exercises should facilitate basic class and sequence diagram understanding and translation to source code. We selected Java as the programming language to match our previous work [3], which we compare as a baseline. In general, the exercise should focus on letting the students explain some parts of a given diagram, extend diagrams with additional predefined elements, and translate parts of the diagram to Java. As ChatGPT offers the upload of files, we distinguish generation without a file input and generation with a class or sequence diagram as the file input.

## A. UML Class Diagram

In the following, we elaborate on our findings for creating basic SE exercises for class diagrams. In our initial prompt, we asked ChatGPT to act as a Software Engineering professor who is teaching a beginner-level to advanced-level SE course that covers modeling and Java. We further stated that the exercise's topic is about *class* diagrams and whether ChatGPT should create its own diagram or whether we provide an initial diagram as Mermaid DSL or graphically. The concrete tasks ChatGPT has to generate should cover an explanation of parts of the diagram, modeling of additional diagram elements that ChatGPT predefines, and translation of diagram parts to Java.

1) *Generation without Diagram Input:* We generated two exercises using the prompt described above and refined it with additional prompts (see Zenodo archive<sup>1</sup>). We generally did not experience significantly better results than we achieved with *GPT-3.5-turbo* [3]. The created class diagrams are not notably more creative, and modeled associations have no roles, descriptions, or multiplicity characteristics. Instead, only basic class diagram features are included. We also noted that the generated class diagram additionally contains associations as attributes. We refined the tasks with additional prompts to extend the generated tasks and add roles to the initial diagram. While extending the tasks worked acceptably in terms of more detailed descriptions and adding additional subtasks, the extension still covered only essential class diagram elements, especially no composition or aggregation. Furthermore, ChatGPT failed to add roles to the initial diagram. Instead, association descriptions were created. Therefore, while the generated exercises are acceptable for a basic exercise sheet, they did not practice more advanced class diagram modeling.

2) *Generation with Textual Diagram Input:* We created two exercises with textual diagram input. As input, we selected the Mermaid DSL version of the *Convoy* and *Ladybug* examples we already used to evaluate ChatGPT's comprehension of class diagrams. With a DSL input, we found that ChatGPT performed slightly better than without an input diagram. The explanation tasks focused especially on the semantics of the diagram and also on the understanding of the DSL's syntax. However, it did not cover roles, descriptions, and multiplicity characteristics for associations. We further noted that the quality of the task to extend the diagram depends on how large the input diagram is. Small diagrams seem to provide fewer clues for ChatGPT to propose an extension with fitting semantics. Nevertheless, ChatGPT can extend especially more complex diagrams by adding additional classes and relations. We further noticed that the translation to the Java task considers classes with their modeled attributes, methods, and relations.

3) *Generation with Graphical Diagram Input:* Similar to the generation with textual diagram input, we used the same two examples for the graphical input. We selected the PNG versions with markable text from the diagrams of Section III. We noticed that when using graphical input, ChatGPT performs slightly better than without input. The generated explanation tasks focus primarily on the semantics of the diagram and do not ask for concrete syntactical elements. ChatGPT

did not cover roles, multiplicity characteristics, or descriptions for associations. Similar to the textual input, extending the diagram depended on how large the input diagram was, and translation tasks also considered relations and not only classes.

## B. UML Sequence Diagram

In the following, we elaborate on our findings for creating basic SE exercises for sequence diagrams. In our initial prompt, we asked ChatGPT to act as a Software Engineering professor who is teaching a beginner-level to advanced-level SE course that covers modeling and Java. We further stated that the exercise's topic is about *sequence* diagrams and stated whether ChatGPT should create its own diagram or whether we provide an initial diagram as Mermaid DSL or graphically. The concrete tasks should cover an explanation of parts of the diagram, modeling of additional elements that ChatGPT predefines, and translation of the sequence to Java, assuming relevant classes and methods already exist.

1) *Generation without Diagram Input:* We generated two exercises without providing a predefined diagram as input. Thereby, we noticed that ChatGPT always created the same example with only slight variants, which was not resolved by regenerating. Especially, the generated diagram did not contain complex sequence diagram elements, e.g., loops or alternatives. Nevertheless, the exercises' quality was acceptable for a starter exercise about sequence diagrams.

2) *Generation with Textual Diagram Input:* We used the sequence diagram examples of Section III as textual diagram input to generate two exercises. Using the DSL input, ChatGPT performed better than without input. The explanation tasks focused on the semantics of the diagram and the DSL syntax. However, ChatGPT had challenges creating a task about extending the diagram as sometimes it lacked finding semantically fitting objects. Nevertheless, extensions that primarily add additional sequences between existing objects with additional messages worked well.

3) *Generation with Graphical Diagram Input:* While ChatGPT covered all relevant elements in the *Ladybug* example's explanation task, the *Convoy* example's explanation task was more general and fuzzy. Additionally, similar to the generation with a textual diagram input, extending the diagram did not work well as sometimes the semantics of added objects were lacking. However, ChatGPT used similar additional classes as those it added to the class diagram generation.

## C. Discussion

Generally, in our experiments, the quality of the generated exercises did not significantly improve compared to *GPT-3.5-turbo* and could be improved. The created exercises primarily use basic diagram elements, and tasks about extending diagrams are uncreative or do not entirely match the initial model's semantics. We found no real difference in the quality of the generated exercise between the graphical diagram and the DSL-based diagram input. For sequence diagrams, providing no input diagram always seems to cover a basic server, client, and database example and, thus, is not fitting for exercise generations that require stronger variability.

## VI. KEY TAKEAWAYS AND THREATS TO VALIDITY

### A. Key Takeaways

We identify four key takeaways. (1) Regarding UML diagram comprehension, ChatGPT mostly understands basic and more advanced diagram elements for textual and graphical diagrams similarly well. It mainly partially understood composition and aggregation and often interpreted aggregations as compositions and vice versa. (2) Generally, ChatGPT could understand the most for textual diagrams. We found that Mermaid-rendered class diagrams result in less accurate comprehension due to their auto-layout and elements hiding others. (3) Regarding the generation of class and sequence diagrams, ChatGPT could not create formally correct diagrams using the underlying *DALL-E*. Instead, the results were more playful than useful. (4) Regarding the generation of modeling exercises, ChatGPT, using a diagram input, slightly outperformed the generation without providing a diagram input. However, we could not identify a solid difference to modeling exercise generation using *GPT-3.5-turbo* from related work.

### B. Threats to Validity

We identify three threats to validity in this study. (1) First, our study is limited to two scenarios for class and sequence diagrams. Other scenarios and other UML diagram types may achieve different results. (2) We did not evaluate the quality of comprehension, UML diagram generation, and exercise design in an actual course, but only by manual review. (3) The textual generation of UML diagrams by ChatGPT can be realized in different modeling languages. While we used *Mermaid*, *PlantUML* and other languages could achieve different results. However, despite the threats mentioned, we consider the findings and insights gathered valuable and beneficial for educators, practitioners, and other researchers.

## VII. RELATED WORK

The related work on this research is rather limited. So far, there is no relevant research on the topic of analyzing UML diagrams. Similarly, there are no research findings on generating graphical UML diagrams with ChatGPT (i.e., the underlying *DALL-E*). When creating textual UML diagrams with DSLs, Abdelfattah et al. [2], Cámará et al. [5] and Fill et al. [6] have shown in their approaches how UML diagrams can be generated with ChatGPT. However, all three papers focus on the generation in *PlantUML* format, Cámará et al.'s generation is carried out with *GPT-3*, and Abdelfattah et al. focus on the generation of UML diagrams by student inputs who require assistance. In previous work [3], [9], we investigated the generation of exercises for programming courses, including modeling exercises with UML using ChatGPT. We also created UML diagrams in the *Mermaid* format for SE exercises [3]. However, this exercise creation was done without provided background and input but focused solely on generating first ideas and scenarios for the exercises.

## VIII. CONCLUSION

In this work, we have investigated ChatGPT's aptitude to deal with UML diagrams for SE education. We investigated the level of information ChatGPT can extract from uploaded UML class and sequence diagrams in different graphical and textual formats and how well ChatGPT can generate UML diagrams independently. Based on the results, we tried to use the information from the UML diagrams as input to create modeling exercises incorporating its specific information. The results show that ChatGPT understands almost all information from textual UML diagrams, while with graphical input, a lot can be detected and interpreted, but not all elements can be understood equally well. The textual generation of UML diagrams worked well, while the graphical modeling of formally correct UML diagrams did not work. We reported our process and the exercise generation and identified good practices and key takeaways. The results of this study provide educators, practitioners, and other researchers a way to create their own exercises, taking into account the strengths and weaknesses described here, to achieve the desired result more efficiently. Nevertheless, a manual review of modeling exercises generated by ChatGPT is indispensable. In future work, we plan to apply different evaluation methods to the exercises generated by ChatGPT to improve the quality of the exercises.

## REFERENCES

- [1] M. Daun and J. Brings, "How ChatGPT Will Change Software Engineering Education," in *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1*, ser. ITICSE 2023. New York, NY, USA: Association for Computing Machinery, 2023, p. 110–116. [Online]. Available: <https://doi.org/10.1145/3587102.3588815>
- [2] A. M. Abdelfattah *et al.*, "Roadmap for Software Engineering Education using ChatGPT," in *2023 International Conference on Artificial Intelligence Science and Applications in Industry and Society (CAISAIS)*, 2023, pp. 1–6. [Online]. Available: <https://doi.org/10.1109/CAISAIS59399.2023.10270477>
- [3] S. Speth, N. Meißner, and S. Becker, "Investigating the Use of AI-Generated Exercises for Beginner and Intermediate Programming Courses: A ChatGPT Case Study," in *2023 IEEE 35th International Conference on Software Engineering Education and Training (CSEE&T)*. [Online]. Available: <https://doi.org/10.1109/CSEET58097.2023.00030>
- [4] B. du Boulay, "Artificial Intelligence as an Effective Classroom Assistant," *IEEE Intelligent Systems*, vol. 31, no. 6, pp. 76–81, 2016. [Online]. Available: <https://doi.org/10.1109/MIS.2016.93>
- [5] J. Cámará, J. Troya, L. Burgueño, and A. Vallecillo, "On the assessment of generative AI in modeling tasks: an experience report with ChatGPT and UML," *Software and Systems Modeling*, vol. 22, no. 3, pp. 781–793, Jun 2023. [Online]. Available: <https://doi.org/10.1007/s10270-023-01105-5>
- [6] H.-G. Fill, P. Fettke, and J. Köpke, "Conceptual modeling and large language models: impressions from first experiments with ChatGPT," *Enterprise Modelling and Information Systems Architectures (EMISAJ)*, vol. 18, pp. 1–15, 2023. [Online]. Available: <https://doi.org/10.18417/emisa.18.3>
- [7] J. D. Velásquez-Henao, C. J. Franco-Cardona, and L. Cadavid-Higuera, "Prompt Engineering: a methodology for optimizing interactions with AI-Language Models in the field of engineering," *DYNA*, vol. 90, no. 230, p. 9–17, Nov. 2023. [Online]. Available: <https://doi.org/10.15446/dyna.v90n230.111700>
- [8] J. White *et al.*, "A Prompt Pattern Catalog to Enhance Prompt Engineering with ChatGPT," 2023. [Online]. Available: <https://doi.org/10.48550/arXiv.2302.11382>
- [9] N. Meißner, S. Speth, and S. Becker, "Automated Programming Exercise Generation in the Era of Large Language Models," in *2024 IEEE 36th International Conference on Software Engineering Education and Training (CSEE&T)*.



# Towards Integrating Emerging AI Applications in SE Education

Michael Vierhauser

Department of Computer Science  
University of Innsbruck  
Innsbruck, Austria  
Michael.Vierhauser@uibk.ac.at

Iris Groher

Institute of Business Informatics –  
Software Engineering  
Johannes Kepler University Linz  
Linz, Austria  
iris.groher@jku.at

Tobias Antensteiner

Clemens Sauerwein  
Department of Computer Science  
University of Innsbruck  
Innsbruck, Austria  
Tobias.Antensteiner@uibk.ac.at  
Clemens.Sauerwein@uibk.ac.at

**Abstract**—Artificial Intelligence (AI) approaches have been incorporated into modern learning environments and software engineering (SE) courses and curricula for several years. However, with the significant rise in popularity of large language models (LLMs) in general, and OpenAI’s LLM-powered chatbot ChatGPT in particular in the last year, educators are faced with rapidly changing classroom environments and disrupted teaching principles. Examples range from programming assignment solutions that are fully generated via ChatGPT, to various forms of cheating during exams. However, despite these negative aspects and emerging challenges, AI tools in general, and LLM applications in particular, can also provide significant opportunities in a wide variety of SE courses, supporting both students and educators in meaningful ways. In this early research paper, we present preliminary results of a systematic analysis of current trends in the area of AI, and how they can be integrated into university-level SE curricula, guidelines, and approaches to support both instructors and learners. We collected both teaching and research papers and analyzed their potential usage in SE education, using the ACM Computer Science Curriculum Guidelines CS2023. As an initial outcome, we discuss a series of opportunities for AI applications and further research areas.

**Index Terms**—AI, Roadmap, Software Engineering Education

## I. INTRODUCTION

Artificial Intelligence (AI) in education, particularly at the university level, has been an active area for several years now, ranging from intelligent tutoring systems to learning analytics support [1], [2]. However, the recent rise in popularity of large language models (LLMs) and chatbots has already disrupted teaching principles in many programming and software engineering (SE) courses. Solutions to programming exercises are automatically generated by students using, for example, ChatGPT and pose further challenges for educators when conducting online exams [3]. Despite the challenges, AI tools in general, and LLM applications in particular, can also be meaningfully employed in a wide variety of SE courses, commonly part of computer science curricula, supporting learners in tasks such as test case generation, architecture pattern creation, or bug detection [4], [5]. Other application scenarios include providing individual feedback to students [6], chatbots to increase learning experience [7], or using AI-based tools as programming assistants [8]. However, so far, only a fraction

of these aspects and novel application areas are covered in SE education and curricula. Moreover, not only learners can benefit from these new opportunities, but also teachers and educators may use them, for example, for generating assignment exercises, or as part of automated grading and assessment systems [9]. In either application scenario, however, the unreflected and careless use of these new tools can also impact the learning experience and outcomes in negative ways, posing new challenges to both students and educators [10]. To exacerbate the situation, existing didactic concepts in the context of SE education hardly take into account or take advantage of these new tools. Also, universities often lack guidelines for the ethical and responsible usage of AI tools in education. Apart from that, students need to be prepared for the ethical implications and societal impacts of AI technologies.

In this paper, we present a preliminary analysis of current trends in the area of AI and how they can potentially be integrated into SE curricula. As part of this, we specifically investigate how they can be leveraged in SE education for both students and educators and how didactic concepts can be adapted to appropriately deal with these new applications. For this purpose, we (1) collected current research on this topic from the past years, (2) performed further snowballing, and (3) grouped the extracted approaches and application areas with regard to their potential usage in SE education, using the ACM Computer Science Curriculum Guidelines version “Gamma”<sup>1</sup>, currently under review.

The main goal of our ongoing work is to gain a better understanding of how the current state of research in AI applications is reflected in SE education, identifying emerging challenges and opportunities for SE courses. As a result, in this short paper, we present an initial overview of techniques and exemplary use cases, alongside a roadmap for further research on and applications of AI tools in SE education.

## II. STUDY DESIGN & RELATED WORK

In this section, we first provide a brief motivation for the need to systematically introduce AI in SE education, and further discuss existing related work in this area.

<sup>1</sup>Please visit csed.acm.org/cs2023-gamma for more information.

### A. Identifying AI Trends in Research and Education

To gain better insights into current trends – particularly within the past year – we started a structured literature analysis for both research and educational venues in software engineering. We specifically chose to include both research and education venues, hoping this will provide a more comprehensive understanding of current research trends and the extent to which they have been integrated into SE education. Furthermore, we aim to document challenges educators need to be aware of and, as a result, potentially adapt didactic concepts and/or teaching materials. In this short paper, we report on our work in progress and the first results of this analysis. For this purpose, we discuss initial findings (cf. Section III) and potential applications and opportunities (cf. Section IV).

To enable a more structured analysis, we further aim to categorize our findings alongside the different SE activities and areas. For SE, numerous taxonomies and classification schemes have been proposed, most notably the Software Engineering Body of Knowledge (SEBoK) [11], with its 18 knowledge areas. As our primary focus, however, is on SE education, we have chosen to group our findings according to the ACM Computer Science Curriculum Guidelines version “Gamma” (CS2023), comprising 17 different knowledge areas, which also incorporate – to a very large extent – the areas proposed in the SEBoK. The ACM Computer Science Curriculum Guidelines are designed to assist educators in the area of computer science, describing the general structure, as well as listing important topics to cover, core and elective CS courses [12]. Most notably, the most recent version, CS2023, currently discussed by the ACM task force, will take into account competency-based education, incorporating a competency model, comprising required skills, and knowledge to be acquired by students. Introducing Learning Analytics and competency-based education in classes typically requires a significant effort and is also something we see huge potential for leveraging AI and alleviating the burden of, for example, manually assessing if all teaching materials correspond to specified competencies (cf. Section IV).

While CS2023 addresses a broad and comprehensive education, covering 17 different knowledge areas, ranging from computer science foundations, to AI, and parallel computing, we focus on two main areas: Software Development Fundamentals (SDF) that cover basic programming education, and second, Software Engineering (SE). A brief overview of the sub-categories for these two areas is shown in Table I. The goal of our work is to identify existing work and approaches for each sub-category of the two areas and collect different means of how AI can be incorporated, for example in teaching aspects of software architecture, such as design patterns, or architectural styles. For this purpose, as a starting point, to gain an initial understanding of current research and trends, we collected an initial set of research papers from 2023, of two prime software engineering conferences: the International Conf. of Software Engineering, and the International Conf. on the Foundations of Software Engineering, as well

as four educational conferences/tracks: Software Engineering Education and Training Track@ICSE, the International Conf. on Software Engineering Education and Training, the Hawaii International Conf. on System Sciences, and the Conf. on Innovation and Technology in Computer Science Education.

The goal was to identify trends and hot topics and uncover areas that might be covered by research but not yet as part of the CS and SE education. In total, we started with a set of 51 publications (33 research-related and 18 education-related), which was extended to 71 after performing an extra round of snowballing. We focused on publications that presented AI applications or educational concepts in the area of SE and programming, but excluded any kind of study, analysis, or review that did not present a specific approach. After carefully reading the titles and abstracts, we excluded 17 papers that were deemed out of scope (e.g. tertiary studies), with a final set of 54 papers remaining.

After selecting our set of papers, we divided them randomly, with four researchers evaluating the papers independently. For the research papers, we focused on the purpose (e.g., test case generation), the technology used, and the specific application area. (Please note, that for the purpose of this paper, and for the sake of brevity, we mainly focus on the intent, and the relation to the CS2023 categories, but future work will broaden the scope of this analysis). For education papers, we also focused on the CS2023 categories, and tried to relate them to the research areas – and ultimately identify gaps and areas that are not yet considered in education. Based on this analysis, we then selected the respective ACM curriculum categories (i.e., Knowledge Units) where the proposed approach, tool, or research could be relevant or applied. After this step, we consolidated all assessments, discussed and resolved conflicts until mutual agreement was achieved, and tried to identify clusters of tools and approaches, as well as gaps. The final goal of the study is to derive a set of concrete actionable guidelines for different courses part of the CS curriculum, with a set of tools and approaches that could be used, and aspects that need to be considered. In the following sections, we discuss the initial assignment of approaches to the CS2023 Knowledge Units and potential application scenarios.

### B. Related Work

Several systematic reviews have investigated and classified aspects of AI applications in education. Bittencourt *et al.* [13] conducted an analysis exploring the intersection of positive psychology and AI in education (P-AIED), identifying P-AIED as a new global movement focusing on positive emotion and engagement related to AI-supported teaching and learning. Kuhail *et al.* [14] and Okonkwo and Ade-Ibijola [15] performed studies related to the application of chatbots in education. Kuhail *et al.* analyzed work on educational chatbots and found that these are primarily used in computer science, language learning, general education, and to a lesser extent in fields like engineering and mathematics. Most chatbots were web-based and more than half functioned as teaching agents, while over a third acted as peer agents. Okonkwo and Ade-

Ibijola analyze 53 primary studies and identify an increasing trend in the application of chatbots in education.

Zhai *et al.* [16] analyzed 100 papers focusing on the application of AI in the education sector from 2010 to 2020. They classified primary studies into a development layer (classification, matching, recommendation, and deep learning), an application layer (feedback, reasoning, and adaptive learning), and an integration layer (affection computing, role-playing, immersive learning, and gamification). Chen *et al.* [17] performed a study investigating the impact of AI on education, focusing on its application and effects in administrative, instructional, and learning contexts. Similarly, Zawacki-Richter *et al.* [18] conducted a systematic review of research on the application of AI in higher education. The majority of research comes from the fields of Computer Science and STEM with AI being mainly used in areas of student profiling and predictive analytics, as well as intelligent tutoring systems.

Several studies have focused on AI in computer science education. Denny *et al.* [19] explore the challenges and opportunities presented by recent advances in AI, particularly code generation models, and their potential impact on computing education. Kalles [20] reports on the experience of using AI systems as the basis of educating IT students. The application areas were mainly decision tree lifecycle management and board game learning mechanisms. Daun and Brings [21] discuss the role of generative AI technologies, such as ChatGPT, in SE education. They discuss potential risks associated with students' use of generative AI but also identify several opportunities that these technologies can offer in enhancing educational practices such as individualization of education and personalized feedback.

Current work in this regard primarily addresses only parts of SE education, for example, programming. With our work, we aim to cover a broader spectrum of SE education, leveraging the ACM Computer Science Curriculum Guidelines.

### III. CURRENT AI TRENDS & APPLICATIONS

The classification of the 54 recent AI papers provides several insights into potential areas where AI topics can be incorporated into computer science and SE courses. Table I provides an overview of the curriculum categories and number of papers relevant to this category, alongside some examples. In the following, we provide a brief summary<sup>2</sup> of potential applications in four distinct categories, which we identified by grouping the categories with the highest coverage.

a) *Introductory Programming Education* (cf. SDF 1 to SDF 4): The integration of AI into courses for novice programmers offers novel methods for enhancing the learning experience and addressing educational challenges. Recent works have studied the application of LLMs to block-based languages [35] which are often used in introductory programming courses. Incorporating AI-based code generation tools [36] can offer real-time coding assistance, thereby facilitating a deeper and more rapid comprehension of programming fundamentals.

Furthermore, pair programming assistants [8] can provide on-demand, context-aware coding suggestions.

b) *Design and Construction* (cf. SDF 4, SE 1, SE 3 to SE 5): Given that software design focuses on the engineering-oriented design of internal software components, "machine-learning-based knowledge extraction techniques" [28] can be employed not only to discover, but also to better understand application programming interfaces (APIs). In the realm of software construction, recent studies have shed light on the robustness of code generation techniques, such as *GitHub Copilot*, which automatically generates code from natural language descriptions [29]. In addition, advances in so-called contrastive learning, exemplified by *CoCoSoDa*, have significantly enhanced the accuracy and performance of code search [30]. Moreover, the emergence of systems such as *AlphaCode* represent a breakthrough as they demonstrate competitive-level code generation capabilities, impacting problem-solving in programming competitions [31].

c) *Maintenance and Testing* (cf. SE 2, SE 7): The integration of AI into more advanced SE courses is currently only sparsely covered in education research, despite the existence of several research advances in this area. This includes, for example, automated test case generation or test augmentation support [37]. LLMs have also been employed in this context [38] fed with bug reports and tasked with generating a series of test candidates. Incorporating these tools in advanced software engineering, and software testing courses can foster awareness of these new tools and technologies. It is, however, also important to note, and convey the message, that these can only serve as an augmentation to existing, traditional, testing methods and are not meant to replace them.

d) *V&V, Reliability, and Security* (cf. SE 6, SE 8, SE 9): The integration of new tools using AI approaches for software verification and validation to recognize bugs or identify vulnerabilities has become an established research area. For example, models such as *CodeBERT* are used to recognize errors in source code [32]. In addition, the combination of ML and NLP has led to the development of novel AI approaches enabling code translation [33], code repair, and inconsistency checking [23] to increase reliability and security of code.

### IV. CHALLENGES & OPPORTUNITIES IN SE EDUCATION

Based on the analysis of trending topics and educational use, we have identified several areas of opportunities where AI could support both educators and students in various tasks. In the following section, we present two directions currently not – or only partially – covered in SE education research, where (1) AI tools should be considered, how AI tools and methods could be applied to (2) alleviate exercise creation as well as grading tasks, and (3) provide tailored support for students. Our literature search has shown that current work related to AI in SE education has a strong focus on exercise generation [39], assessment [9], and grading [6] using LLMs. However, the quality of the results varies and often does not yield satisfactory programming tasks or grading results (cf.

<sup>2</sup>Please refer to doi.org/10.5281/zenodo.11316028 for the full classification.

Table I: ACM CS2023 categories of the two selected knowledge areas SDF and SE (cf. Section II) with examples from our literature search and coverage in research/education (●=Covered in several publications; ○=Partially Covered; ○=Not/Sparsely Covered), and the number of papers (#) with research (R) and education (E) focus.

ID	Category	Description/Example	# (R/E)	Co.
<i>Software Development Fundamentals (SDF)</i>				
SDF 1	Fundamental Programming Concepts and Practices	Basic programming concepts, in conjunction with algorithms and data structures, can be supported by a variety of emerging approaches. Particularly, LLMs can be employed for many different tasks and support roles such as the generation of programming assignments [22], serving as chatbots and interactive tutors [21].	16 (2/14)	●
SDF 2	Fundamental Data Structures		13 (0/13)	●
SDF 3	Algorithms		11 (0/11)	●
SDF 4	Software Development Practices	AI approaches can guide and improve development practices, e.g., by generating or analyzing commit messages [23] or explaining code snippets [24].	8 (5/3)	○
<i>Software Engineering (SE)</i>				
SE 1	Teamwork	AI approaches supporting collaborative work, e.g., for generating commit messages [25].	1 (1/0)	○
SE 2	Tools and Environments	AI-supported SE tools, e.g., chatbots supporting novice developers [26].	9 (6/3)	●
SE 3	Product Requirements	AI-supported question answering tools help analyze and understand NL requirements [27].	1 (1/0)	○
SE 4	Software Design	AI techniques, including knowledge extraction for APIs [28], code generation [29], and advances code search accuracy [30], and systems like AlphaCode [31].	2 (2/0)	○
SE 5	Software Construction		17 (11/6)	●
SE 6	Software Verification and Validation	Novel tools leveraging AI approaches to detect bugs, e.g., using CodeBERT [32].	24 (22/2)	●
SE 7	Refactoring and Code Evolution	AI techniques for code evolution, e.g., for multi-language applications/systems [33].	9 (9/0)	●
SE 8	Software Reliability	ML & NLP methods for code translation, inconsistency detection, and remediation.	20 (19/1)	●
SE 9	Formal Methods	While only sparsely covered so far, LLMs can also be used for, e.g. proof-generation [34].	1 (1/0)	○

Direction I). Furthermore, going beyond introduction to programming courses, i.e., CS1, as well as basic data structures courses, i.e., CS2, we observed, that while being a very active research topic, only a few papers address more advanced areas in SE education (cf. Direction II).

a) *Direction I – Competency-focused Exercise Generation (Educators)*: Following the trend of employing generative AI for a slew of different tasks, one potential application area could be the creation of more diverse and comprehensive exercises, e.g., for programming classes. While similar works have been proposed recently [39], [40], we think that this can go far beyond “simply” creating exercise sheets or programming tasks. Particularly taking into account competency-based education [41], which is also reflected in the most recent 2023 ACM curriculum guidelines, with a dedicated competency model, LLMs could be used in this context. First, to create a set of tasks that covers certain competencies, e.g., specified in a competency model, but also to generate specific test cases that cover/check the achievement of certain competencies, and even provide support for assigning assignments and identifying gaps in missing competences or tasks that exceed what students should be able to accomplish as part of an assignment.

The challenge hereby resides in providing the LLM with the necessary information – e.g., a competency model – and the context and scope of the assignment and tasks.

The benefits are on the one hand a significant reduction in the manual tedious effort of creating variations of exercises (e.g., programming tasks) and on the other hand a move towards competency-based assessment and Learning Analytics support, without the need to manually incorporate these aspects in exercises and assignments.

b) *Direction II – AI in SE (Educators and Students)*: Various authors have stressed the application of AI tools in Software Development Fundamentals (SDF 1, SDF 2, and SDF 3), both as an opportunity and a challenge (e.g., students generating assignments without critical reflection). However,

the deeper application of AI tools in advanced SE classes does not seem to be widely considered [42]. In contrast, we discovered a wide variety of AI applications in research that intersect with basic and advanced software engineering education. Particularly software quality assurance, maintenance, and validation (SE 6 and SE 8) are covered by several research papers, but are hardly present in the educational context.

The challenge from our perspective is striking a balance between those new technologies and tools that alleviate certain tasks and still teaching basic and advanced concepts and theories to students. For example, if code generation tools take over tasks, such as generating architectural patterns or unit tests, it is still crucial to convey knowledge about these foundational aspects so that students know how to apply them properly and understand the concepts behind them.

The benefits are that students are introduced to new tools and paradigms early on in their education that they are very likely to encounter later on during their professional careers. Moreover, it fosters understanding of AI capabilities and limitations, preparing students to understand ethical implications and societal impacts with a well-informed, critical perspective.

## V. OUTLOOK AND CONCLUSION

With the release of ChatGPT, AI tools, and particularly LLMs have significantly impacted both educators and students, often being perceived as a boon and a bane at the same time. While providing a vast amount of opportunities for new and improved tools for generating code, analyzing bugs, or improving code quality, these aspects are only sparsely converted in current software engineering courses. In this early research paper, we present a preliminary analysis of current trends in the area of AI, and how they can be integrated into SE curricula. We further discuss several challenges and opportunities where AI-powered tools and techniques can support teachers and learners, but also need to be taken into account when designing exercises or course assignments.

We will further explore the development and evaluation of didactic methods that integrate AI tools into university-level courses. Additionally, there is a need to investigate the long-term impacts of AI integration on learning outcomes and employability of students.

## REFERENCES

- [1] S. Z. Salas-Pilco, K. Xiao, and X. Hu, "Artificial intelligence and learning analytics in teacher education: A systematic review," *Education Sciences*, vol. 12, no. 8, p. 569, 2022.
- [2] N.-T. Le, S. Strickroth, S. Gross, and N. Pinkwart, "A review of ai-supported tutoring approaches for learning programming," *Advanced computational methods for knowledge engineering*, pp. 267–279, 2013.
- [3] M. M. Rahman and Y. Watanobe, "ChatGPT for education and research: Opportunities, threats, and strategies," *Applied Sciences*, vol. 13, no. 9, p. 5783, 2023.
- [4] M. Schäfer, S. Nadi, A. Eghbali, and F. Tip, "An empirical evaluation of using large language models for automated unit test generation," *IEEE Transactions on Software Engineering*, 2023.
- [5] C. S. Xia, Y. Wei, and L. Zhang, "Automated program repair in the era of large pre-trained language models," in *Proc. of the 45th Int'l Conf. on Software Engineering*, 2023, pp. 1482–1494.
- [6] R. Balse, B. Valaboju, S. Singhal, J. M. Warriem, and P. Prasad, "Investigating the potential of GPT-3 in providing feedback for programming assessments," in *Proc. of the 2023 Conf. on Innovation and Technology in Computer Science Education*. ACM, 2023, p. 292–298.
- [7] D. Benner, S. Schöbel, A. Janson, and J. M. Leimeister, "Engaging minds - how gamified chatbots can support and motivate learners in digital education," in *Proc. of the 57th Hawaii Int'l Conf. on System Sciences*, T. X. Bui, Ed. ScholarSpace, 2024, pp. 54–63.
- [8] A. Moradi Dakhel, V. Majdinasab, A. Nikanjam, F. Khomh, M. C. Desmarais, and Z. M. J. Jiang, "GitHub copilot AI pair programmer: Asset or liability?" *Journal of Systems and Software*, vol. 203, 2023.
- [9] V. González-Calatayud, P. Prendes-Espinosa, and R. Roig-Vila, "Artificial intelligence for student assessment: A systematic review," *Applied Sciences*, vol. 11, no. 12, p. 5467, 2021.
- [10] J. Bergström, V. Repo, and V. Tuunainen, "Challenges to academic integrity from new tools—a survey of students' perceptions and behaviors of employing chatgpt," 2024.
- [11] IEEE Computer Society, "SWEBOK Guide Version 4.0 beta," [https://sebokwiki.org/wiki/SWEBOK\\_Guide\\_Version\\_4.0](https://sebokwiki.org/wiki/SWEBOK_Guide_Version_4.0), Last accessed 01-06-2024.
- [12] R. K. Raj and A. N. Kumar, "Toward computer science curricular guidelines 2023 (cs2023)," *Inroads*, vol. 13, no. 4, pp. 22–25, 2022.
- [13] I. I. Bittencourt, G. Chalco, J. Santos, S. Fernandes, J. Silva, N. Batista, C. Hutz, and S. Isotani, "Positive artificial intelligence in education (PAIEd): A roadmap," *International Journal of Artificial Intelligence in Education*, pp. 1–61, 2023.
- [14] M. A. Kuhail, N. Alturki, S. Alramlawi, and K. Alhejori, "Interacting with educational chatbots: A systematic review," *Education and Information Technologies*, vol. 28, no. 1, pp. 973–1018, 2023.
- [15] C. W. Okonkwo and A. Ade-Ibijola, "Chatbots applications in education: A systematic review," *Computers and Education: Artificial Intelligence*, vol. 2, p. 100033, 2021.
- [16] X. Zhai, X. Chu, C. S. Chai, M. S. Y. Jong, A. Istenic, M. Spector, J.-B. Liu, J. Yuan, and Y. Li, "A review of artificial intelligence (AI) in education from 2010 to 2020," *Complexity*, vol. 2021, pp. 1–18, 2021.
- [17] L. Chen, P. Chen, and Z. Lin, "Artificial intelligence in education: A review," *IEEE Access*, vol. 8, pp. 75 264–75 278, 2020.
- [18] O. Zawacki-Richter, V. I. Marín, M. Bond, and F. Gouverneur, "Systematic review of research on artificial intelligence applications in higher education—where are the educators?" *Int'l Journal of Educational Technology in Higher Education*, vol. 16, no. 1, pp. 1–27, 2019.
- [19] P. Denny, J. Prather, B. A. Becker, J. Finnie-Ansley, A. Hellas, J. Leinonen, A. Luxton-Reilly, B. N. Reeves, E. A. Santos, and S. Sarsa, "Computing education in the era of generative AI," 2023.
- [20] D. Kalles, "Artificial intelligence meets software engineering in computing education," in *Proc. of the 9th Hellenic Conf. on Artificial Intelligence*. USA: ACM, 2016.
- [21] M. Daun and J. Brings, "How chatgpt will change software engineering education," in *Proc. of the 2023 Conf. on Innovation and Technology in Computer Science Education V.1*, 2023, pp. 110–116.
- [22] B. P. Cipriano and P. Alves, "Gpt-3 vs object oriented programming assignments: An experience report," in *2023 Conf. on Innovation and Technology in Computer Science Education V.1*, 2023, pp. 61–67.
- [23] I. Bouzenia and M. Pradel, "When to say what: Learning to find condition-message inconsistencies," in *Proc. of the 45th Int'l Conf. on Software Engineering*. IEEE, 2023, pp. 868–880.
- [24] J. Leinonen, P. Denny, S. MacNeil, S. Sarsa, S. Bernstein, J. Kim, A. Tran, and A. Hellas, "Comparing code explanations created by students and large language models," in *2023 Conf. on Innovation and Technology in Computer Science Education V.1*. ACM, 2023.
- [25] J. Dong, Y. Lou, D. Hao, and L. Tan, "Revisiting learning-based commit message generation," in *Proc. of the 45th Int'l Conference on Software Engineering*, ser. ICSE '23. IEEE Press, 2023, p. 794–805.
- [26] A. Shau, Y. Liang, W. Hsieh, X. Lin, and S. Ma, "PSAbot: A chatbot system for the analysis of posts on stack overflow," in *Proc. of the 35th Int'l Conf. on Software Engineering Education and Training*. IEEE, 2023, pp. 137–141.
- [27] S. Ezzini, S. Abualhaija, C. Arora, and M. Sabetzadeh, "AI-based question answering assistance for analyzing natural-language requirements," *arXiv preprint arXiv:2302.04793*, 2023.
- [28] D. Nam, B. Myers, B. Vasilescu, and V. Hellendoorn, "Improving API knowledge discovery with ML: A case study of comparable API methods," in *45th Int'l Conf. on Software Engineering*, 2023, pp. 1890–1906.
- [29] A. Mastropaoletti, L. Pasquarella, E. Guglielmi, M. Ciniselli, S. Scalabrinio, R. Oliveto, and G. Bavota, "On the robustness of code generation techniques: An empirical study on github copilot," in *Proc. of the 45th Int'l Conf. on Software Engineering*, 2023, pp. 2149–2160.
- [30] E. Shi, Y. Wang, W. Gu, L. Du, H. Zhang, S. Han, D. Zhang, and H. Sun, "CoCoSoDa: Effective contrastive learning for code search," in *Proc. of the 45th Int'l Conf. on Software Engineering*, 2023, pp. 2198–2210.
- [31] Y. Li, D. Choi, J. Chung, N. Kushman, J. Schrittweiser, R. Leblond, T. Eccles, J. Keeling, F. Gimeno, A. Dal Lago *et al.*, "Competition-level code generation with alphacode," *Science*, vol. 378, no. 6624, pp. 1092–1097, 2022.
- [32] A. Grishina, M. Hort, and L. Moonen, "The earlybird catches the bug: On exploiting early layers of encoder models for more efficient code classification," *arXiv preprint arXiv:2305.04940*, 2023.
- [33] J. Zhang, P. Nie, J. J. Li, and M. Gligoric, "Multilingual code co-evolution using large language models," in *Proc. of the 31st ACM Joint European Software Engineering Conf. and Symposium on the Foundations of Software Engineering*, 2023, pp. 695–707.
- [34] E. First, M. Rabe, T. Ringer, and Y. Brun, "Baldur: Whole-proof generation and repair with large language models," in *Proc. of the 31st ACM Joint European Software Engineering Conf. and Symposium on the Foundations of Software Engineering*, 2023, pp. 1229–1241.
- [35] E. Griebl, B. Fein, F. Obermüller, G. Fraser, and R. Just, "On the applicability of language models to block-based programs," in *Proc. of the 45th Int'l Conf. on Software Engineering*, 2023, pp. 2374–2386.
- [36] B. Reeves, S. Sarsa, J. Prather, P. Denny, B. A. Becker, A. Hellas, B. Kimmel, G. Powell, and J. Leinonen, "Evaluating the performance of code generation models for solving parsons problems with small prompt variations," in *Proc. of the 2023 Conf. on Innovation and Technology in Computer Science Education V.1*. ACM, 2023, p. 299–305.
- [37] P. Nie, R. Banerjee, J. J. Li, R. J. Mooney, and M. Gligoric, "Learning deep semantics for test completion," *preprint - arXiv:2302.10166*, 2023.
- [38] S. Kang, J. Yoon, and S. Yoo, "Large language models are few-shot testers: Exploring LLM-based general bug reproduction," in *Proc. of the 45th Int'l Conf. on Software Engineering*. IEEE, 2023, pp. 2312–2323.
- [39] S. Speth, N. Meißner, and S. Becker, "Investigating the use of ai-generated exercises for beginner and intermediate programming courses: A ChatGPT case study," in *Proc. of the 35th Int'l Conf. on Software Engineering Education and Training*. IEEE, 2023, pp. 142–146.
- [40] T. C. Freitas, A. Costa Neto, M. J. V. Pereira, and P. R. Henriques, "NLP/AI based techniques for programming exercises generation," in *Proc. of the 4th Int'l Computer Programming Education Conf.* Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2023.
- [41] M. C. Pluff and V. Weiss, "Competency-based education: The future of higher education," *New Models of Higher Education: Unbundled, Rebundled, Customized, and DIY*, pp. 200–218, 2022.
- [42] M. Johnson, "Generative AI and CS education," *Communications of the ACM*, vol. 67, no. 4, p. 23–24, Mar. 2024.



# Toward AI-facilitated Learning Cycle in Integration Course through Pair Programming with AI Agents

Zhengyuan Wei, Albert T.L. Lee, Victor C.S. Lee  
*Department of Electrical and Electronic Engineering  
The University of Hong Kong  
Pokfulam, Hong Kong  
zywei4@hku.hk, {tlalee, csvlee}@eee.hku.hk*

Wing-Kwong Chan  
*Department of Computer Science  
City University of Hong Kong  
Kowloon Tong, Hong Kong  
wkchan@cityu.edu.hk*

**Abstract**—We propose a new methodology that harnesses recent advancements in AI techniques to formulate an AI-facilitating code learning cycle for students. The approach builds on an existing learning process and innovatively incorporates pair programming into the learning cycle. It first transforms the example code into scaffold code as exercises through an instructor-AI pairing. The scaffold code serves as an exercise for students to complete and debug on a hardware platform iteratively with an expert AI assistant. This design alleviates instructors' burden of crafting new exercises for new scenarios and offers students the advantage of interactive learning with scenario diversity. We evaluate the methodology using a suite of example codes and assess the semantic similarity among different code versions produced by AI assistants. The case study shows promising results of the methodology. We further discuss our findings and outline future work for the proposed methodology.

**Index Terms**—Generative AI, pair programming, learning process

## I. OVERVIEW

Generative Artificial Intelligence (GenAI) demonstrates its potential to generate high-quality content with distilled knowledge encoded in large language models [1]. ChatGPT [2], a prominent GenAI tool, has gained considerable attention. Many universities have deployed their hosted versions of ChatGPT, enabling students and instructors to utilize the GenAI tools in their teaching and learning activities. Nevertheless, despite GenAI's effectiveness in creating diverse content, its role in facilitating the teaching and learning process for students and educators remains poorly understood.

In this study, we propose an innovative approach to establish an interactive learning cycle, utilizing AI models to enhance students' learning experience in an Electrical and Electronic Engineering (EEE) discipline core course focused on programming with hardware modules in our department. Prior to this method, the existing learning process illustrated in Fig. 1 was narrow, as the students primarily relied on the materials provided by our instructor  $\mathcal{I}$  to explore (e.g., reading reference materials) and incorporate new modules into the course materials. These materials encompassed example code  $C^*$  (see example in Listing 1) that had been thoroughly tested and confirmed to function seamlessly on a target hardware platform  $H$  (e.g., Arduino). Although students observed the effects of running  $C^*$  on  $H$ , they could only adopt the instructor's code and did not engage in any programming practice when learning to interact with the hardware module.



Fig. 1. Existing learning process of an EEE integration course.

```

1  /* Simple program that blinks an LED. */
2  const int ledPin = 13;
3
4  void setup() {
5      pinMode(ledPin, OUTPUT);
6  }
7
8  void loop() {
9      digitalWrite(ledPin, HIGH);
10     delay(1000);
11     digitalWrite(ledPin, LOW);
12     delay(1000);
13 }
```

Listing 1. Arduino example code used in the course

Programming demands practical applications of knowledge [3], and completing exercises plays a crucial role in fostering an effective acquisition of cognitive skills [4]. However, creating code versions  $C^\bullet$ , equivalent to  $C^*$  from  $\mathcal{I}$ , solely by students can be challenging, as it often requires consulting the technical hardware manuals and expert knowledge to identify relevant information manually, which are slow and challenging particular to beginners or weaker students. In other words, developing such code from scratch without any assistance from the instructor can be a daunting task. One potential solution to support students in producing  $C^\bullet$  is to provide a platform-styled structured code skeleton with essential and informative comments as an initial code version for students to start with (see example in Listing 2). In this study, we refer to such code as *scaffold code* (denoted as  $C^\circ$ ) throughout the paper.

On the other hand, interactive learning is an educational method focused on student engagement and grounded in the philosophy of constructive alignment [5]. Starting with the modification of the scaffold code, students can practice programming by independently completing  $C^\circ$  and debugging it based on the observed effects on  $H$ . This approach offers a more interactive learning and deeper experience compared to

the traditional process, where students actively engage in the implementation and debugging of  $C^*$  on H in our proposed learning cycle (refer to Fig. 2 for a visual representation).

While the proposed learning cycle presents an ideal scenario, it poses two potential practical challenges. First, crafting detailed scaffold code from scratch can be time-consuming for instructors. Second, it is uncertain whether the scaffold code can lead to a code version (referred to as *complement code*)  $C^*$  equivalent to  $C^*$  in behavior. Furthermore, students may not always produce a workable  $C^*$  that functions as expected on the hardware platform. For these reasons, we are developing a novel methodology to tackle these challenges to support our novel interactive learning cycle. One key area is to what extent the code version  $C^*$  can indeed be equivalent to  $C^*$  in behavior after modification if students are *zero knowledge*. As such, assessing the semantic similarity between  $C^*$  produced by an AI assistant and  $C^*$  is a critical step in assuring the feasibility of our proposed methodology.

The main contribution of this paper is as follows: It presents our novel methodology that creates a new interactive learning cycle. To our knowledge, this work is the *first work* to propose an AI-facilitated learning cycle that employs pair programming with AI agents as solutions to tackle the challenges.

The rest of the paper is organized as follows. We present the preliminaries, our methodology and learning cycle, and the evaluation in §II, §III, and §IV. §V discusses the lessons learnt and related work. §VI concludes the paper.

## II. PRELIMINARIES

### A. Large Language Models, Prompts, and Context

Large language models (LLMs) [6] are artificial neural networks with hundreds of billions of parameters notable for their ability to achieve outstanding language generation. The inputs to the LLMs are generally texts written in natural language, also known as *prompts*, containing any of the following elements: instruction, input data, and context. Instruction is a specific task or request the user wants the model to perform. Input data contains content for LLMs to read and generate from. *Context* is external information or additional examples that can steer the LLMs to better outputs. For instance, a reference code can serve as an example to indicate the LLMs to generate outputs in the same format.

### B. Pair Programming

Pair programming [7] is a software development technique in which two programmers work together on a task. Between them, one programmer is called *driver* that writes code, and the other is called *navigator* that reviews the code. The navigator is more responsible for the code quality. According to the knowledge level of the programmers, there are two roles of partners, producing three pairing variations [8]. The professional partner(s) is deemed as *expert*, while the inexperienced partner(s) is deemed as *novice*. Expert-novice is the most conventional pairing that allows the expert to mentor the novice. Expert-expert may seem to be the obvious pairing for the highest productivity and can produce great results.

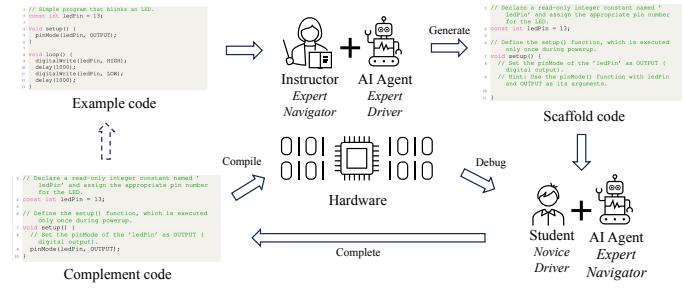


Fig. 2. Overview of AI-facilitated learning cycle with pair programming

## III. METHODOLOGY

Fig. 2 depicts our proposed learning cycle. To address the challenges mentioned in Section I, LLMs are introduced as AI agents to pair with humans to perform the tasks in the cycle, in which the AI agents take different roles in different pairs.

### A. Scaffold Code Generation

This methodology M assumes the example code  $C^*$  in the existing learning process is available to the instructor  $\mathcal{I}$ . It pairs  $\mathcal{I}$  with an AI agent  $\mathcal{A}_1$  to construct an expert-expert pairing, where  $\mathcal{A}_1$  is the driver to generate a scaffold code  $C^*$  based on  $C^*$ , and  $\mathcal{I}$ , as navigator, inspects the generated  $C^*$ .

M requires a human test oracle  $\mathcal{O}$  from  $\mathcal{I}$ . A test oracle [9] is a mechanism to determine the correctness of the code under test.  $\mathcal{I}$  ensures the instruction correctness of  $C^*$  by assessing the comments in  $C^*$  to the code statement in  $C^*$ . The assessment operation is denoted by applying  $\mathcal{O}$ .

In this way, the time-consuming issue can be addressed by adopting an efficient AI agent, and the instructor can guard the code quality of  $C^*$  to avoid misleading students to produce a complement code  $C^*$ .

### B. Interactive Code Complement

Given the scaffold code  $C^*$  from  $\mathcal{I}$ , students try their best to complete  $C^*$  by themselves and debug the code interactively with the hardware platform H. With the effects and console outputs from H, students can debug and recompile the code to H iteratively, producing a complement code version  $C^*$  finally. Students gain more programming practices, benefitting from interactive learning.

In case some students cannot produce  $C^*$  in the above process for some exercises, M ensures  $C^*$  by allowing the usage of LLMs in these exercises. M pairs the student  $\mathcal{S}$  with an AI agent  $\mathcal{A}_2$  in expert-novice pattern to produce  $C^*$ .  $\mathcal{S}$  as the driver completes  $C^*$ , while  $\mathcal{A}_2$  assists  $\mathcal{S}$  in completion and debugging as the navigator.(An example AI coding assistant could be found in [10].)

In this way, students would always produce a complement code version  $C^*$  workable on H guaranteed by  $\mathcal{A}_2$ , as if they obtain an example code in the traditional process. They can also benefit from pair programming with the AI agent.

### C. Semantic Assessment

The feasibility of the proposed methodology heavily depends on the semantics between  $C^*$  and  $C^\circ$ . M provides two types of assessment for the semantic difference.

1. *Automatic*: Moss [11] (for a Measure Of Software Similarity) is a widely-used automatic system in academics for determining the similarity of two pieces of code developed by Stanford University. It also notes a misuse of Moss if relying solely on the similarity score.

2. *Manual*: To complement the Moss system, we apply  $\mathcal{O}(C^*, C^\circ)$  to mute the false alarm of Moss in the evaluation.

M reports the semantic similarity score from Moss and the semantical equivalence from  $\mathcal{O}$  in the evaluation.

## IV. CASE STUDY

We implement the methodology as an open-source tool. The repository includes the evaluation dataset and the assessment reports at <https://github.com/Wsine/pair-prgm-learn>.

### A. Background

Our case study adopts the teaching materials from the EEE course *Integrated Design Project* offered by our department. This course teaches students programming with hardware modules on the Arduino platform. The evaluation dataset  $\mathcal{D}$  is compiled with all 32 pieces of Arduino example code used in the course. As the impact of the proposed learning cycle on students is still uncertain, we have not involved any students in this case study. The purpose of this case study is to evaluate the feasibility of the methodology and the new learning cycle.

### B. Generating AI-assisted Scaffold Code

In this step, we adopt ChatGPT 4 [2] as  $\mathcal{A}_1$  and develop a useful prompt to instruct  $\mathcal{A}_1$  to generate the scaffold code.

#### Prompt to Generate Scaffold Code

Below is an Arduino code. Please remove all the comments first. Then, generate detailed comments for each statement with many hints for students to fill in. Last, remove all the code statements in function bodies but keep all the comments and the function signatures. Output the formatted final code only.

```
```cpp
{{ example_code }}
```

```

#### Final example code for reference

```
```cpp
Scaffold code template as context to guide generation
```

```

The prompt is developed in markdown format to maintain the semantics for different text sections. Listing 1 shows a piece of example code  $C^*$  in the course. We input  $C^*$  in the

```

1 /* Simple program that blinks an LED. */
2
3 // Declare a read-only integer constant named 'ledPin' and assign the appropriate pin number
4 // for the LED.
5 const int ledPin = 13;
6
7 // Define the setup() function, which is executed
8 // only once during powerup.
9 void setup() {
10    // Set the pinMode of the 'ledPin' as OUTPUT (
11    // digital output).
12    // Hint: Use the pinMode() function with ledPin
13    // and OUTPUT as its arguments.
14 }
15
16 // Define the loop() function, which is called
17 // repeatedly during the program execution.
18 void loop() {
19    // Turn the LED on by setting a HIGH voltage to
20    // the 'ledPin'.
21    // Hint: Use the digitalWrite() function with
22    // ledPin and HIGH as its arguments.
23
24    // Wait for 1000 milliseconds (1 second) before
25    // executing the next command.
26    // Hint: Use the delay() function with 1000 as
27    // its argument.
28 }

```

Listing 2. Arduino scaffold code generated by ChatGPT

$\{\{example\_code\}\}$  section as input data and consistently use a short scaffold code example (akin to Listing 2) in the last section as context to guide the generation.  $\mathcal{A}_1$  produces the corresponding scaffold code  $C^\circ$  exhibited in Listing 2.

As we can interpret from the code listing, scaffold code  $C^\circ$  maintains the semantics of the program from the code statements in  $C^*$  to the instructive comments in  $C^\circ$ . Each key statement in  $C^*$  is transformed into a pair of instruction and hint comments in  $C^\circ$ . Students can follow the comments in  $C^\circ$  to reproduce the code business logics of  $C^*$ , during which they practice programming.

We repeat the above procedure to the remaining example code in  $\mathcal{D}$  to generate their associated scaffold code.

### C. Completing Code with AI Agents

The most widely adopted AI assistant agent is GitHub Copilot [10]. Since we do not involve students in this case study, we mimic the expert-assisted code completion process by leveraging ChatGPT 4 [2] as  $\mathcal{A}_2$ . Note that GitHub Copilot also uses ChatGPT 4 as its underlying AI model.

We develop a prompt to instruct  $\mathcal{A}_2$  to produce the complement code  $C^*$  in this case study. We input the scaffold code

```

1 /* Simple program that blinks an LED. */
2
3 // Declare a read-only integer constant named 'ledPin' and assign the appropriate pin number
4 // for the LED.
5 const int ledPin = 13;
6
7 // Define the setup() function, which is executed
8 // only once during powerup.
9 void setup() {
10     // Set the pinMode of the 'ledPin' as OUTPUT (digital output).
11     pinMode(ledPin, OUTPUT);
12 }
13
14 // Define the loop() function, which is called
15 // repeatedly during the program execution.
16 void loop() {
17     // Turn the LED on by setting a HIGH voltage to
18     // the 'ledPin'.
19     digitalWrite(ledPin, HIGH);
20     // Wait for 1000 milliseconds (1 second) before
21     // executing the next command.
22     delay(1000);
23     // Turn the LED off by setting a LOW voltage to
24     // the 'ledPin'.
25     digitalWrite(ledPin, LOW);
26     // Wait for 1000 milliseconds (1 second) before
27     // executing the next command.
28     delay(1000);
29 }
```

Listing 3. Arduino complement code generated by ChatGPT

$C^o$  generated in the last step to the  $\{\{\text{scaffold\_code}\}\}$  section of the prompt.  $A_2$  generates  $C^*$  as a result.

#### Prompt to Generate Complement Code

Below is Arduino code with mostly comments. Please understand the business logic according to the comments and then complete the code. Output the formatted final code only.

```
```cpp
{{ scaffold_code }}
```

Listing 3 shows the generated complement code  $C^*$  corresponding to  $C^o$  in Listing 2 and  $C^*$  in Listing 1. Comparing  $C^*$  with  $C^*$ , the code statements (ignoring the comments) in the two pieces of code are identical. It implies that LLMs have the capability to create exercises for students and assist students in producing semantically equivalent code to run on the hardware platform. Students can benefit from the assistance of AI agents if pair programming is adopted in their learning.

We repeat the above procedure to all scaffold code produced in the last step to generate their associated complement code.

#### D. Semantic Evaluation

The evaluation dataset consists of 32 pieces of code, covering diverse hardware modules and code complexity. Table I describes the statistics of the evaluation dataset. Each test case

TABLE I  
STATISTICS AND RESULTS OF EVALUATION DATASET

Code #	Brief description	LoC	Moss	$\mathcal{O}(C^*, C^*)$
C1	1MHz square wave	20	⊖	✓
C2	Vanilla motor control	97	38%	✓
C3	Motor control with encoder	68	67%	✓
C4	Motor control with timer	80	43%	
C5	Monitor Gyro with MPU	34	97%	✓
C6	Bi-directional power monitor	131	⊖	✓
C7	Communicate with LoRa	38	93%	✓
C8	OLED display	81	49%	✓
C9	Handle PWM signal	33	30%	✓
C10	WiFi connection	31	93%	✓
C11	MQTT receiver	58	78%	✓
C12	MQTT sender	50	98%	✓
C13	MQTT receiver with timer	74	68%	✓
C14	MQTT sender periodically	59	57%	✓
C15	MQTT with QTI	106	88%	✓
C16	MQTT with OLED	137	41%	✓
C17	MQTT with LoRa	65	66%	✓
C18	Signal with timer	115	77%	✓
C19	LED blink	16	96%	✓
C20	Basic button	16	93%	✓
C21	Button interruption	29	95%	✓
C22	Button polling	27	91%	✓
C23	Basic buzzer	10	91%	✓
C24	LED fade	25	53%	✓
C25	Light tracking	102	82%	✓
C26	OLED simple	87	98%	✓
C27	QTI simple	27	54%	✓
C28	Servo delay control	27	98%	✓
C29	Servo periodical control	33	97%	✓
C30	Basic timer	59	92%	✓
C31	Ultrasonic delay measure	32	95%	✓
C32	Ultrasonic time measure	43	81%	✓

#### Moss Results

Thu Mar 7 04:58:24 PST 2024

Options -l cc -m 10

[ [How to Read the Results](#) | [Tips](#) | [FAQ](#) | [Contact](#) | [Submission Scripts](#) | [Credits](#) ]

File 1	File 2	Lines Matched
<a href="#">example/blink.ino (96%)</a>	<a href="#">complementement/blink.ino (96%)</a>	21

Any errors encountered during this query are listed below.

Fig. 3. Test report returned by Moss

(i.e., example code  $C^*$ ) is named after  $C_i$  in the first column, followed by a brief description of its code, the number of lines of code (LoC), the semantic similarity score from Moss, and the manual test, respectively.

We collect all pairs of  $(C^*, C^*)$  generated from the previous two steps. Then, we feed all the pairs separately into the Moss system [11] and acquire the Moss test reports. Fig. 3 and Fig. 4 demonstrate a test report pertaining to the test case in Listing 1. If similarity is detected, it reports the results on the file list, the semantic similarity score between  $(C^*, C^*)$  in the parentheses, and the highlighted code difference in another webpage. We extract all the similarity scores and report them in Table I. The symbol  $\ominus$  denotes no similarity detection in the Moss

```

example/blink.ino (96%)
5-15
complementement/blink.ino (96%)
6-26

```

```

example/blink.ino

/*
Simple program that blinks an LED.
Reference: ELEC3848 Lab 1
*/
const int ledPin = 13;           // declare read-only integer ledPin, set to correct pin number
void setup() {                  // the setup() method is executed only once during powerup
    pinMode(ledPin, OUTPUT);     // the ledPin PIN is declared as digital output
}
void loop() {                   // the loop() method is called repeatedly
    digitalWrite(ledPin, HIGH);   // set a HIGH voltage to ledPin PIN
    delay(1000);                // stop the program for 1000 milliseconds
    digitalWrite(ledPin, LOW);    // set 0V (ground) to ledPin PIN
    delay(1000);                // stop the program for 1000 milliseconds
}

complementement/blink.ino

/*
Simple program that blinks an LED.
*/
// Declare a read-only integer constant named 'ledPin' and assign the appropriate pin number for the LED.

const int ledPin = 13;           // Define the setup() function, which is executed only once during powerup.
void setup() {                  // Set the pinMode of the 'ledPin' as OUTPUT (digital output).
    pinMode(ledPin, OUTPUT);
}
// Define the loop() function, which is called repeatedly during the program execution.
void loop() {                   // Turn the LED on by setting a HIGH voltage to the 'ledPin'.
    digitalWrite(ledPin, HIGH);   // Wait for 1000 milliseconds (1 second) before executing the next command.
    delay(1000);                // Turn the LED off by setting a LOW voltage to the 'ledPin'.
    digitalWrite(ledPin, LOW);    // Wait for 1000 milliseconds (1 second) before executing the next command.
    delay(1000);
}

```

Fig. 4. Highlighted code difference returned by Moss

test reports. We can observe that only two test cases (i.e.,  $C1$  and  $C6$ ) do not report any similarity, and 25/32 test cases (i.e., 78%) have detected higher than the 50% similarity score.

Further, we carefully compare each ( $C^*$ ,  $C^\bullet$ ) and check the semantics difference. The results are reported in the last column of Table I, where the symbol  $\checkmark$  stands for *semantical equivalence*. Most of the pairs of ( $C^*$ ,  $C^\bullet$ ) are semantically equivalent, except for only one test case (i.e.,  $C4$ ). We have the following findings.

1) Different parameters may be generated, resulting in a slight semantic shift. Taking  $C4$  as an example, `analogWrite(9, 150)` about motor controlling is generated in  $C^\bullet$  compared with `analogWrite(9, 255)` in  $C^*$ . This kind of shift does not affect students studying programming but provides diverse parameters for exploration.

2) The Moss system does not count for similarities in global comments and printing functions. This contributes to  $C6$  and most test cases with the deduction of semantic similarity scores and no similarity detection in the worst case.

3) Different implementations may be generated, but the semantics are the same. For instance, in  $C1$ , `bit (COM1A0)` is used in  $C^*$  while `(1 << COM1A0)` is implemented in  $C^\bullet$ .

## V. DISCUSSION AND RELATED WORK

We observe no information loss in the cycle in which LLMs cast the code statements in  $C^*$  into instructive comments in  $C^\circ$  and then back to the code statements in  $C^\bullet$ . Educators can use this observation to generalize to other courses.

The significance of our methodology is rooted in the opportunities it provides for practicing programming. Ebbinghaus's forgetting curve indicates that rapid learning, such as merely reading, is often followed by rapid forgetting [12]. A follow-up study reveals that 40%–60% of freshly acquired knowledge may be lost within a day without practice [13].

Many studies have explored the impact of GenAI on education. For example, the works [14], [15] explored creating challenging exercises for ChatGPT to encourage students' critical

thinking. Additionally, Microsoft Research [16] presents studies on the usability and design of AI-assisted programming, further demonstrating the positive impact of AI on education.

## VI. SUMMARY AND FUTURE WORK

Our study proposes an AI-facilitated learning cycle for both teaching and learning with a novel methodology to address the practical challenges in the cycle. The evaluation showcases promising results of the methodology. Students gain interactive learning in the programming practices compared to the existing learning process. In the future, we plan to incorporate the learning cycle and methodology into courses and refine the approach to enhance students' learning processes further.

## REFERENCES

- [1] L. Banh and G. Strobel, "Generative artificial intelligence," *Electron Markets*, 2023.
- [2] OpenAI. (2024 Mar 7 version) Chatgpt 4. <https://chat.openai.com/chat>.
- [3] T. M. Connolly, M. Stansfield, and T. Hainey, "An application of games-based learning within software engineering," *Brit. J. of Edu. Tech.*, 2007.
- [4] K. VanLehn, "Cognitive skill acquisition," *Annu. Rev. of Psychol.*, 1996.
- [5] G. E. Dames, "Enhancing of teaching and learning through constructive alignment," *Acta Theologica*, 2012.
- [6] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," 2019.
- [7] L. Williams, "Integrating pair programming into a software development process," in *Proc. 14th Conf. on Softw. Eng. Edu. and Train.*, 2001.
- [8] L. Williams and R. R. Kessler, *Pair programming illuminated*, 2003.
- [9] W. Howden, "Theoretical and empirical studies of program testing," *IEEE Tran. on Softw. Eng.*, 1978.
- [10] GitHub. Copilot. <https://github.com/features/copilot>.
- [11] A. Aiken. (2024 Mar 7 version) Moss. <https://theory.stanford.edu/~aiken/moss/>.
- [12] H. Ebbinghaus, "Memory: A contribution to experimental psychology," *Ann. Neurosci.*, 2013.
- [13] J. Murre and J. Dros, "Replication and analysis of ebbinghaus' forgetting curve," *PloS One*, 2015.
- [14] W. Chan, Y. Yu, W. Keung, and C. Lee, "Toward AI-assisted exercise creation for first course in programming through adversarial examples of ai models," in *CSEE&T'23*, 2023.
- [15] J. Berrezueta-Guzman and S. Krusche, "Recommendations to create programming exercises to overcome chatgpt," in *CSEE&T'23*, 2023.
- [16] A. Sarkar, A. Gordon, C. Negreanu, C. Poelitz, S. Ragavan, and B. Zorn, "What is it like to program with artificial intelligence?" *arXiv*, 2022.



# University Students' Perception and Expectations of Generative AI Tools for Software Engineering

Mounika Yabaku

Dept. Information Technology  
Uppsala University  
Uppsala, Sweden  
mounika.thandlam@gmail.com

Sofia Ouhbi

Dept. Information Technology  
Uppsala University  
Uppsala, Sweden  
sofia.ouhbi@it.uu.se

**Abstract**—Adopting Generative Artificial Intelligence (AI) tools in software engineering represents a shift in how tasks like coding and idea generation are approached. This paper investigates university students' perceptions and expectations regarding the use of Generative AI tools such as ChatGPT and Copilot in software engineering. To achieve this, we conducted a questionnaire study with volunteer participants studying at Uppsala University in Sweden, resulting in a total of 127 responses. These responses were about the usage preferences, motivations for adoption, perceived benefits, encountered challenges, and suggested improvement of these tools. The findings reveal that 16% of participants have never used a Generative AI tool, while of those who have used such tools predominantly use ChatGPT3.5. Among users of Generative AI, respondents reported benefits such as code optimization and idea generation, alongside challenges such as inaccuracies in generated content and understanding user intent. Despite these challenges, participants perceive the integration of Generative AI tools as transformative for traditional software engineering practices. The results of this paper offer insights into the practical use of AI tools and suggestions for improving their functionality, thereby influencing the future direction of software engineering education.

**Index Terms**—Generative AI, software engineering, university students, perception, expectation, questionnaire.

## I. INTRODUCTION

Software engineering is facing several challenges due to the rising complexity and evolution of software systems [1]. These challenges include the pressing need for enhanced automation, innovation, and efficiency in software development processes, coupled with the dynamic and diverse nature of software requirements [2]. Generative Artificial Intelligence (AI) emerges as a promising solution to these challenges. Generative AI encompasses diverse methods that enable systems to generate new outputs, often drawing from patterns or examples in previously collected data. This capability positions Generative AI as a promising tool to tackle these issues, creating new and realistic material from existing data sources such as text, graphics, and code [3].

The integration of Generative AI tools into software engineering marks a transformative era with the potential to significantly reshape both processes and outcomes of software engineering practices [4]. These technologies promise to make software development more accessible to individuals with varying technical proficiencies, democratizing software creation, fostering innovation and empowering problem-solving

[4]. However, alongside their promising applications, Generative AI technologies present limitations such as potential inaccuracies or biases in generated data, which could impact software reliability or security [5]. Additionally, ethical, legal, and social considerations may arise in their deployment [6]. Therefore, software engineers need to exercise caution and rigorously verify and test data and code before integration into their projects.

Numerous studies have investigated the use of Generative AI tools in software engineering. Giriprasad Sridhara et al. [7] discussed ChatGPT's applications in tasks such as ambiguity resolution, method naming, code review, and log summarization, noting limitations in testing and debugging due to ChatGPT's inaccuracies. Yetistiren et al. [8] explored various code generation technologies in AI-assisted tools. Ebert and Louridas [9] detailed Generative AI's productivity boosts and software integration. Sida Peng et al. [10] examined GitHub Copilot's impact on professionals' efficiency. Ozkaya and Ipek [11] emphasized the role of proper automation in successful software development. Russo [12] highlighted AI technology adoption among engineers. Gerhard et al. [13] analyzed Generative Design and Engineering's implications for software development. Bull et al. [14] provided insights into the use of generative AI in software development education, suggesting pedagogical improvements.

Despite the increasing prevalence of Generative AI tools in software engineering, their features and user expectations remain largely unexplored, particularly in university settings. This paper aims to bridge this gap by offering insights into the perceptions and expectations of university students regarding these tools. The study presents the results of an online questionnaire aimed at understanding future software engineers' views on the use of Generative AI tools in software engineering.

## II. METHOD

A self-administered online questionnaire, developed using Google Forms, was distributed to university students at Uppsala University in Sweden. Participants were recruited from social network groups of university students focused on areas related to software engineering, human-computer interaction, and computer science at Uppsala University. The

TABLE I  
QUESTIONNAIRE)

Questions	Response options
<b>Section 1</b>	
Level of education Have you used any Generative AI tools for software development (requirements, designing, coding, testing, etc.)? If No, please elaborate why you never use this?	Bachelor's, Master's, PHD, Other Yes, No Open question
<b>Section 2</b>	
Have you used any of the following Generative AI tools?  If so, how often? Why are you using Generative AI tools for software engineering tasks?  What aspects of Generative AI tools do you appreciate the most for software engineering? Have you improved your skills in using Generative AI tools? Have these tools changed the way you approach software engineering? Have you encountered any limitations or challenges while using these tools? If yes, please specify the challenges What improvements would you suggest for these tools?  How comfortable would you be using a tool to automatically generate entire functions or code blocks? How do you think these tools will impact the field of software engineering in the long term? What are your expectations or preferences for the future development of Generative AI tools?  How would you rate your overall experience with these tools? Briefly describe your overall experience here. Would you recommend these tools to your friends/colleagues?	GPT 3.5, bard, Copilot, GPT 4, Replit, Hugging Face, BlackBox, Replicate, TabNine, Cody, Scraper, AskCodi, Browse AI Daily, Weekly, Monthly, Rarely Code optimization, Automated testing, Reduction of human error, Generation of new ideas or solutions, Keeping up with the latest technology trends, Documentation, Image generation, Other Accuracy, Performance, Compatibility, Ease of use, Other Yes, regularly, Yes, occasionally, No, but I plan to in the future, No, and I have no plans to, Other Yes, significantly, Yes, somewhat, No change, Not applicable (haven't used Generative AI tools) Yes, frequently, Yes, occasionally, No, never, Not applicable (haven't used Generative AI tools) Open question Accuracy in outputs, More comprehensive documentation, Improved performance (faster response times, etc.), More user-friendly interfaces, Support for multiple functionalities, Online Browsing capacity, Freely available, Other Very comfortable, Somewhat comfortable, Neutral, Somewhat uncomfortable, Very uncomfortable Increase Efficiency, Improved Quality, Enhanced creativity, Change in Development Process, Threatening job offers, Other More advanced capabilities, Better user experience, Greater integration with other tools, More robust security features to protect sensitive data, More customization options, Expansion of AI capabilities beyond current limitations, Other Very positive, Somewhat positive, Neutral, Somewhat negative, Very negative Open question Yes, No, Maybe

questionnaire description clearly stated that participation was voluntary, anonymous, and could be withdrawn at any time. No incentives were provided for participation in this study.

Table I provides an overview of the questionnaire, which included a mix of open-ended and closed-ended questions. The questionnaire comprised two sections: the participants' level of study and prior experience with Generative AI, and their perceptions and expectations regarding Generative AI for software engineering practices. Participants were only required to answer the first two questions regarding their education level and prior use of Generative AI, while all other questions were optional. If a participant indicates that they have never used Generative AI tools, Section 2 of the questionnaire is automatically skipped after providing a reason why they never used such tools, and a thank you message for their participation is displayed.

The questionnaire was designed with ethical considerations in mind, ensuring no collection of sensitive data. Prior to the main study, the survey underwent testing by three researchers to ensure clarity. The questionnaire was accessible for responses from February 27th to March 18th, 2024.

### III. RESULTS

A total of 127 university students participated in this questionnaire study. The results are available in this shared

document: <https://shorturl.at/bsvOP>.

#### A. Section 1

Table II presents the results of Section 1 of the questionnaire.

TABLE II  
STUDY PARTICIPANTS AND THE TOOL USAGE (N=127)

Education Level	Total (%)
Bachelor's	47 (37%)
Master's	67 (53%)
PhD	10 (8%)
Other	3 (2%)
Using any tool?	Total (%)
Yes	107 (84%)
No	20 (16%)

Out of the 127 participants, 37% held Bachelor's degrees, 53% held Master's degrees, 8% held Ph.D. degrees, and 2% had other qualifications, specifying system engineering. These results highlight the diverse range of student qualifications represented in the study. Among the participants, 84% reported using Generative AI tools for software development tasks such as requirements, design, coding, debugging, and testing, while 16% (20 out of 127) reported not using any Generative AI tools in their work. Only those participants who reported using these

tools, totaling 107 participants, proceeded to Section 2 of the questionnaire.

Participants who did not use Generative AI tools prior to responding to the questionnaire provided a diverse range of reasons. These include viewing the tools as unnecessary for their work or learning goals, preferring personal skills and alternatives like Stack Overflow. Some cited a lack of involvement in AI-focused tasks, doubts about AI's effectiveness for complex work, and concerns about producing incorrect code. Additionally, there were expressions of disinterest, mistrust leading to unsatisfactory outcomes, and a desire for independent learning. Participants also mentioned skepticism about AI's industry hype, disagreement with its politics, and criticism of biases in AI models. Ethical concerns about labor and creativity, a preference for hands-on problem-solving, and doubts about AI's applicability to their field were noted. Issues of security, trust, dislike for AI methods, and its potential to stifle creativity or coding independence were also raised. Concerns were also reported about the reliability of AI-generated code, tech industry hype influencing AI adoption, and broader societal ethical implications.

## B. Section 2

The most used Generative AI tools are summarized in Figure 1. ChatGPT3.5 is predominant across educational backgrounds, with 101 out of 107 respondents using it. Copilot is next most used tool with 47 users, primarily among master's students ( $n=31$ ), bachelor's students ( $n=10$ ), PhD students ( $n=4$ ), and others ( $n=2$ ). GPT-4 follows with 46 users, mainly among master's students ( $n=29$ ), bachelor's students ( $n=14$ ), PhD students ( $n=2$ ), and one with other qualifications. Bard is utilized by 28 respondents, including 20 master's students, 6 bachelor's students, and 2 PhD students. Similarly, BrowseAI is reported to be used by 17 respondents, with a notable preference among bachelor's students ( $n=8$ ), alongside 7 master's students and 1 PhD student. Lastly, Blackbox is used by 15 respondents, predominantly among 11 master's students and 4 bachelor's students.

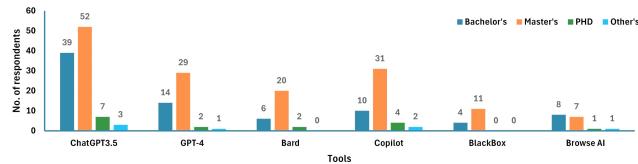


Fig. 1. Most used tool

Figure 2 illustrates the motivations behind individuals' use of Generative AI tools in their daily activities. The data reveals that majority of respondents utilize these tools to generate novel ideas and solutions. A large number of respondents reported that they used these tools for code optimization as a key factor, while some reported using them to reduce human errors, maintain and generate documentation, automate testing, and stay current with technology trends. A smaller percentage of users utilize AI tools for image creation. The remaining

categories, such as information retrieval, code generation, debugging, handling repetitive tasks, and generating fake data for prototypes, account for a few responses collectively.

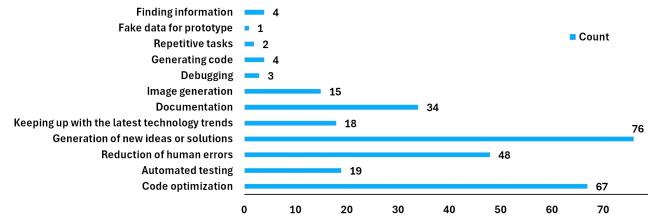


Fig. 2. Purpose of using Generative AI

Figure 3 presents the features that software engineers most appreciate in Generative AI tools. The largest percentage of respondents value ease of use as the most important aspect. Following closely, participants liked the accuracy and performance of Generative AI tools. Another significant factor is compatibility with existing software and systems. A few respondents noted the importance of technical discussions and guidelines.

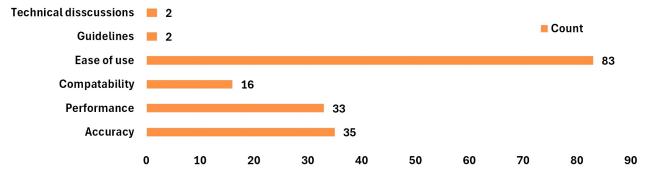


Fig. 3. Most liked features of Generative AI tools

Figure 4 illustrates the impact of professional development on skills, encountered limitations, and changes in software development. Regarding improved skills with Generative AI tools, 54 individuals occasionally use Generative AI tools, with 31 using them regularly. Thirteen plan to use these tools in the future, while seven do not intend to start. Regarding changes in software engineering approach: 59 participants noted a somewhat altered approach, while 29 respondents saw a significant change in their approach due to Generative AI technologies. 15 participants stated no change, and 3 did not use Generative AI tools for tasks related to software engineering. Regarding encountered limitations or challenges, 45 participants reported frequent challenges, with 49 facing occasional issues. 13 participants had not experience any limitations.

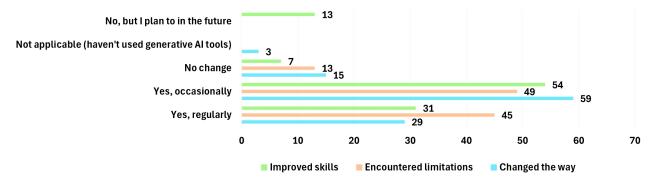


Fig. 4. Impact on professional development

Figure 5 shows expectations for the future development of Generative AI tools. Among the provided responses: expecting

advanced capabilities, anticipating seamless integration with other software development tools, prioritizing a better user experience, expecting enhanced security measures to safeguard confidential data, foreseeing expanded AI capabilities beyond current limitations, and seeking customizable tools to meet specific requirements.



Fig. 5. Participants' expectations

Figure 6 illustrates the long-term impact of Generative AI tools on software engineering. Respondents anticipate increased efficiency, a shift in development methods, quality improvement, and concerns about job security. One respondent mentioned concerns about output quality, and another highlighted employers' expectations for increased work with minimal staff.



Fig. 6. Perceived long-term Impact of Generative AI tools

Figure 7 shows suggested improvements for existing Generative AI tools. The majority of respondents ( $n=93$ ) emphasized the need for improved output accuracy. Prioritizing free availability was important for a large number of participants ( $n=47$ ), while around the third of participants highlighted the necessity of comprehensive manuals and tutorials for effective tool usage. Some participants mentioned the desire for expanded functionalities, noted the importance of enhancing tool performance, and expressed interest in improved online browsing capabilities, such as better integration with online resources or cloud-based functionalities. Only a few participants called for a more user-friendly interface.

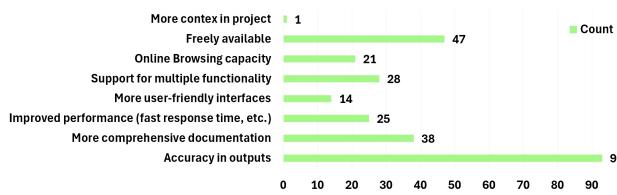


Fig. 7. Suggested improvements

The majority of participants reported frequent or occasional limitations when using Generative AI tools. Respondents reported that they often need to verify AI-generated content

for accuracy, which may not align with their coding style or preferences. Inaccurate information can persist, highlighting the reliance on the quality and quantity of training data. AI's understanding of user intent or requirements may also be incomplete, resulting in unclear responses. Furthermore, AI's contextual comprehension of queries may be limited.

Figure 8 represents the overall experience and recommendation of these tools. The majority of participant ( $n=62$ ) were somewhat positive, while 32 participants expressed a very positive sentiment towards these tools, and 13 remained neutral. When asked about the likelihood of recommending Generative AI tools to peers and acquaintances, almost half of the participants said they would recommend them, only one participant who was neutral about these tools reported not wanting to recommend them.

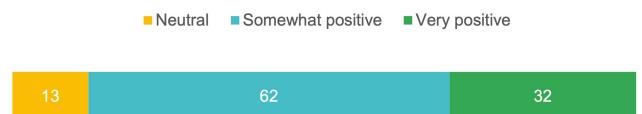


Fig. 8. Overall experience using Generative AI tools

Participants described AI technologies as significantly increasing productivity, with positive outcomes experienced across various tasks. However, some mentioned encountering issues with coding and finding certain areas, such as math, challenging. Responses also indicated a varied experience with AI tools, sometimes providing good results and at other times leading to frustration and stress. It was noted that using AI tools requires a careful approach and a degree of caution to ensure optimal results. Overall, the sentiment towards Generative AI tools was largely positive, with users finding them beneficial for generating ideas and describing them as fantastic tools for various purposes.

#### IV. DISCUSSION

The main findings of this study are listed below.

a) *High Adoption of Generative AI Tools:* The study reveals a substantial adoption rate of 84% among university students in software engineering. This trend suggests a growing interest and reliance on AI-driven solutions within the student community. The reasons behind this surge in adoption likely stem from the perceived advantages of automation, heightened productivity, and exposure to cutting-edge technologies [15].

b) *Preference for ChatGPT3.5:* Among the Generative AI tools surveyed, ChatGPT3.5 stands out as the primary choice across diverse educational backgrounds. Factors contributing to its popularity include free accessibility, user-friendly interface, and ability to produce contextually relevant outputs. Its general application in aiding tasks such as coding, debugging, and documentation are listed among the reason why university students prefer it [16].

c) *Reasons for Hesitancy in Using Generative AI Tools:* Some participants expressed reservations about using Generative AI tools, citing reasons such as a perceived lack of

necessity in their work, a preference for self-reliance, or ethical concerns surrounding these technologies. This emphasizes the importance of understanding individual attitudes and perceptions towards AI in software engineering. Addressing these concerns through clear communication of the benefits and potential applications of Generative AI tools could potentially boost their adoption rate [17].

*d) Benefits and Challenges of Generative AI Tools:* Users reported numerous benefits from employing Generative AI tools, such as generating novel ideas, code optimization, and error reduction. These advantages align with AI's potential in automating routine tasks and enhancing overall productivity. However, users also faced challenges, including inaccuracies in generated content, difficulties in deciphering user intent, and limitations in handling complex tasks. Addressing these challenges is critical for refining user satisfaction and tool effectiveness.

*e) Impact on Software Engineering Approach:* The survey indicates that Generative AI tools have somewhat altered the approach of many users towards software engineering. This suggests that AI technologies are reshaping conventional workflows and methodologies by introducing opportunities for automation, optimization, and innovative problem-solving. As AI continues to evolve, it is poised to become an integral component of software engineering practices, influencing how developers design, implement, and maintain software systems.

*f) Expectations for Future Development:* Users articulated diverse expectations regarding the future development of Generative AI tools, including the desire for advanced capabilities, enhanced user experiences, and seamless integration with existing tools. Fulfilling these expectations can significantly contribute to the continued adoption and efficacy of AI in software engineering. Moreover, users emphasized the importance of output accuracy, comprehensive documentation, and improved performance in future iterations of AI tools. Meeting these demands is imperative for fostering trust and confidence among users.

## V. CONCLUSION

Based on the findings of this study, it is evident that university students in software engineering are increasingly reliant on Generative AI tools. The data revealed a substantial 84% of students utilizing these tools for a range of development tasks, with ChatGPT3.5 emerging as the favoured option. While these tools offer the potential to enhance productivity and encourage innovation, users face challenges such as inaccuracies and the inherent difficulty of Generative AI in interpreting user intent. Addressing these obstacles is crucial to enhancing user satisfaction and the overall efficacy of Generative AI tools. The results highlight a critical need for future research to focus on refining these tools to improve their effectiveness, enhancing user experience, and addressing ethical considerations for responsible integration within software engineering practices.

Considering the high adoption rate of Generative AI tools among university students, academia should adapt its curriculum to include comprehensive training on these tools. Offering

courses or workshops focused on the practical applications and ethical considerations of AI in software engineering can prepare students for the evolving industry landscape. Additionally, fostering research collaborations with industry partners can provide students with hands-on experience and real-world projects involving Generative AI tools. This interdisciplinary approach will not only enhance students' skill sets but also equip them with the knowledge and tools needed to navigate the growing AI-driven landscape in software engineering.

## ACKNOWLEDGMENT

We would like to express our gratitude to all the participants who actively took part in this study. We also extend our special thanks to Venugopal Thandlam for his valuable suggestions.

## REFERENCES

- [1] S. Ouhbi and N. Pombo, "Software engineering education: Challenges and perspectives," in *2020 IEEE Global Engineering Education Conference (EDUCON)*. IEEE, 2020, pp. 202–209.
- [2] P. Bourque and R. E. Fairley, "Guide to the software engineering body of knowledge, version 3.0," 2014, [www.swbok.org](http://www.swbok.org).
- [3] S. Feuerriegel, J. Hartmann, C. Janiesch, and P. Zschech, "Generative AI," *Business and Information Systems Engineering*, vol. 66, no. 1, p. 111–126, Sep. 2023.
- [4] C. B. Frey and M. Osborne, "Generative AI and the future of work: a reappraisal," *Brown Journal of World Affairs*, pp. 1–12, 2023.
- [5] H. Tian, W. Lu, T. O. Li, X. Tang, S.-C. Cheung, J. Klein, and T. F. Bissyandé, "Is ChatGPT the ultimate programming assistant—how far is it?" *arXiv preprint arXiv:2304.11938*, 2023.
- [6] G. Lawton, "Generative AI ethics: 8 biggest concerns and risks," Nov 2023. [Online]. Available: <https://www.techtarget.com/searchenterpriseai/tip/Generative-AI-ethics-8-biggest-concerns>
- [7] G. Sridhara, R. H. G., and S. Mazumdar, "ChatGPT: A study on its utility for ubiquitous software engineering tasks," *arXiv preprint arXiv:2305.16837*, 2023.
- [8] B. Yetişiren, I. Özsoy, M. Ayerdem, and E. Tüzin, "Evaluating the code quality of AI-assisted code generation tools: An empirical study on GitHub Copilot, Amazon CodeWhisperer, and ChatGPT," 04 2023.
- [9] C. Ebert and P. Louridas, "Generative AI for software practitioners," *IEEE Software*, vol. 40, no. 4, pp. 30–38, 2023.
- [10] S. Peng, E. Kalliamvakou, P. Cihon, and M. Demirer, "The impact of AI on developer productivity: Evidence from GitHub Copilot," *arXiv preprint arXiv:2302.06590*, 2023.
- [11] Ozkaya and Ipek, "The next frontier in software development: AI-augmented software development processes," *IEEE Software*, vol. 40, no. 4, pp. 4–9, 2023.
- [12] D. Russo, "Navigating the complexity of generative AI adoption in software engineering," *arXiv preprint arXiv:2307.06081*, 2023.
- [13] D. Gerhard, T. Köring, and M. Neges, "Generative engineering and design – a comparison of different approaches to utilize artificial intelligence in cad software tools," in *Product Lifecycle Management. PLM in Transition Times: The Place of Humans and Transformative Technologies*, F. Noël, F. Nyffenegger, L. Rivest, and A. Bouras, Eds. Cham: Springer Nature Switzerland, 2023, pp. 206–215.
- [14] C. Bull and A. Kharrufa, "Generative AI Assistants in Software Development Education: A vision for integrating Generative AI into educational practice, not instinctively defending against it." *IEEE Software*, 2023.
- [15] C. Zastudil, M. Rogalska, C. Kapp, J. Vaughn, and S. MacNeil, "Generative AI in computing education: Perspectives of students and instructors," in *2023 IEEE Frontiers in Education Conference (FIE)*, 2023, pp. 1–9.
- [16] C. K. Y. Chan and W. Hu, "Students' voices on generative AI: Perceptions, benefits, and challenges in higher education," *International Journal of Educational Technology in Higher Education*, vol. 20, no. 1, p. 43, 2023.
- [17] K. Kenthapadi, H. Lakkaraju, and N. Rajani, "Generative AI meets responsible AI: Practical challenges and opportunities," in *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2023, pp. 5805–5806.



# A Deep Dive Into CATS Evaluator Algorithm: Quantification Of The Probability in Serious Game Cloud Security Defense Scenarios

Tiange Zhao  
and Didem Ongu  
and Tiago Gasiba  
Siemens AG  
Otto-Hahn-Ring 6  
81739, München, Germany  
Email: tiange.zhao@siemens.com  
Email: didem.oengue@siemens.com  
Email: tiago.gasiba@siemens.com

Ulrike Lechner  
University of the Bundeswehr Munich  
Werner-Heisenberg-Weg 39  
85579, Neubiberg, Germany  
Email: ulrike.lechner@unibw.de

Maria Pinto-Albuquerque  
Instituto Universitário de Lisboa  
(ISCTE-IUL)  
ISTAR, Av. das Forças Armadas  
1649–026 Lisboa, Portugal  
Email: maria.albuquerque@iscte-iul.pt

**Abstract**—Cloud deployment has become increasingly common due to its flexibility and business value. However, cloud assets face cybersecurity challenges and need to be configured securely. Industry practitioners must be trained to understand key concepts in cloud security, including ‘defense & attack’ and ‘roles & responsibilities.’ A serious game provides an engaging and helpful way to convey such messages. This work introduces the core evaluator algorithm developed for Cloud of Asset and Threats (CATS), a serious game designed to enhance cloud security awareness.

This work builds upon our previous efforts, focusing on refining the Evaluator algorithm that quantifies the probabilities of the defender strategies as defined by the players to prevent a given attack vector from being successful. We present the results collected from industry training events where the refined algorithm was incorporated into CATS and compare them to the initial implementation. The promising results indicate that with the refinement of the evaluator algorithm, more elements derived from reality are addressed, and the game maintains a similar difficulty level for participants.

## I. INTRODUCTION

Cloud deployment has become a mature business with increasing connectivity and efficiency. Applying cloud services reduces operators’ maintenance workload and provides access to professional management also of the security of applications.

This paper presents the refinement of the Cloud Asset and Threats (CATS) serious game. CATS is a tabletop game played in training in the industry to enhance cloud security. Specifically, our focus is to improve the evaluator algorithm applied in the simulation to calculate the probability of a defense strategy created by the participants successfully blocking a certain attack approach. The plausibility and transparency of the evaluator are important as we would like to simulate real-world cloud security issues. A systematic approach to creating the evaluator is necessary to introduce additional threat scenarios and adapt the game to new threats and technological advances.

This article extends previous work on the design of the CATS game [1], [2], [3] and presents evaluations of the game in industrial training. It is essential to note that the participants in our study were industry practitioners in various industry departments, including IoT devices, healthcare, and energy.

The paper is structured as follows: Section II presents important related work as a foundation of our study. Section III outlines our study methodology. Section IV does the deep dive into CATS background and the evaluator algorithm. Section V illustrates the evaluation process and results. Finally, Section VI concludes our work with a discussion of next steps.

## II. RELATED WORK

In this section, we present the related work for our research, which consists of IT security awareness, serious games in cybersecurity, and cloud security as a foundation of this paper.

### A. *IT security awareness*

The work of Hänsch et al. on IT security awareness [4] defines three dimensions of IT security awareness: perception, protection, and behavior. Perception is related to knowing about IT security, protection is related to knowing how to protect IT assets, and behavior is related to the intention to actively protect the IT assets.

### B. *Serious games in cybersecurity*

Serious games are games with purposes other than pure entertainment as defined in the field of serious game design by Dörner et al. [5]. Our work follows and extends this baseline.

There are various serious games that aim to raise cybersecurity awareness. Shostack lists security tabletop games on his website [6]. An industry example is Riskio by Hart et al. [7], designed for non-technical individuals in organizations, effectively increasing cybersecurity awareness. Unlike Riskio,

CATS targets industrial practitioners focusing on cloud security concepts.

Another Week at the Office (AWATO) [8] by Ferro et al., based on a systematic literature review, focuses on the human factor and phishing attacks. The evaluation shows it is an effective tool for improving users' cybersecurity awareness. This work supports the idea that serious games can address the awareness issue. However, CATS stands out as one of the first serious games dedicated to cloud security, employing an automatic evaluator algorithm to simulate defense and attacks strategically, providing a reliable probability result.

### C. Cloud security

The Cloud Controls Matrix (CCM) [9] from the Cloud Security Alliance (CSA) provides a well-established mapping and comparison of various security controls and cross-checks 44 relevant industrial standards. Notable among these are ISO 27017 and 27018 [10], [11], which specifically address information security controls for cloud services. Apart from the mentioned standards, the MITRE ATT&CK Cloud Matrix categorizes possible attacks against cloud systems based on real-world hacking activities [12]. For each potential attack, effective defense mechanisms are listed. These standards mandate the implementation of a secure development process and provide specific security requirements for cloud security in the industry. This useful information is conveyed to the practitioners in the industry through training where serious games are helpful enhancing methods for raising awareness about cybersecurity issues and solutions.

In our work, the game elements are based on the mapping proposed by the MITRE ATT&CK Cloud Matrix. Additionally, we consider different roles and responsibilities in defending cloud assets. The attack scenarios are abstracted from real-world hacking activities against known cloud assets.

## III. METHODOLOGY

This work follows Hevner et al.'s design science paradigm [13], focusing on the Design & Implement and Justify & Evaluate cycle.

In the current design iteration, the focus is on the refinement of the evaluator algorithm and systematic mappings with a common vulnerability scoring system (CVSS), and common vulnerabilities and exposures (CVE) that occurred in real-world cloud security incidents. This design iteration is strategically chosen to address identified limitations and continuously improve the serious game. The goal in this iteration is to enhance the realism of attack scenarios and improve the evaluator algorithm to provide players with a better understanding of the consequences of their defense strategies without significantly increasing the game difficulty. Our work contributes to the continuous improvement of the serious game and enhances cloud security awareness among industry practitioners.

The refined evaluator algorithm is tested in 2 game events with 24 industrial practitioners. This targeted testing allows for a focused assessment of the improvements. The evaluation

aims to verify the feasibility of the refined evaluator algorithm and measure the performance of the players compared to the previous design.

## IV. ALGORITHM DESIGN

In this section, we first present background information about the CATS game, where the evaluator algorithm operates as a core element. Then, we provide technical aspects of the algorithm's design details. To give a straightforward impression, in the last part of this section, we go through the algorithm step-by-step with a concrete example.

### A. Background information about CATS game

CATS features six attack scenarios based on real-world incidents, taking approximately 60 minutes to complete. Each scenario follows a three-step kill chain: Initial Access (2 actions), Launch Attack (3 actions), and Make an Impact (1 action). In the current CATS version, the players can utilize the available defense cards and arrange a defense plan to defend the could attack against a given attack scenario.

To defend against a specific attack, players must choose six defense action cards from 23 available cards and assign them to the correct responsibilities. CATS has two responsibilities: business and technical. The former represents business decisions needed for implementing a certain defense (e.g., determining account use policies), while the latter signifies technical implementation tasks (e.g., configuring network segmentation). If a selected card is assigned to the wrong responsibility, it is discarded during the evaluation process and does not contribute to the success of the defense plan. Players can assign a maximum of two cards to business responsibility and four to technical responsibility to strategize the defense plan.

The defense plan and the attack scenario serve as inputs to the evaluator algorithm, producing a probability indicating the likelihood of the defense withstanding the attack. For each attack scenario, a threshold is assigned. If the algorithm's output is higher than the threshold, with all defense cards correctly placed and all attack actions mitigated by at least one defense card, the player successfully solves the scenario and can progress to the next one. If the output probability is lower than the threshold or other conditions are not met, the evaluator proposes hints, allowing the player to adjust the defense plan. There is no limit to the number of trials, and players are encouraged to experiment with different combinations and learn about cloud security.

In this work, we focus on a refinement of the mapping of defense and attack actions by introducing the CVSS scoring system into the conditional probability calculation.

### B. Impact of the defense actions

There is a many-to-many mapping between defense and attack actions. When a defense action mitigates a vulnerability, the defense action has an impact on its CVSS vector.

In CATS, the base metric group of CVSS vectors is used. We apply CVSS 3.1 [14], as it is the mainstream version of CVSS for the time being.

A valid defense has two different types of impact upon an attack. It could either reduce the factor by one value or eliminate the potential impact on a certain vector of the system or component.

Let's assume there is an attack action Network Service Discovery [15] mapped to the CVE cve-2020-1206 [16]. The vector in the base metric of cve-2020-1206 is:

CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N

with a score of 7.5 (High). We apply the defense action Network Segmentation, the impacts of which are to reduce attack vector (AV) and scope (S) by one value. AV is updated from Network (N) to Adjacent (A). Since the Scope of the original CVE is Unchanged (U), which is already the lowest value and cannot be further reduced, the second impact does not have any substantial influence on the given attack action. By applying Network Segmentation, the vulnerability's vector is updated to:

CVSS:3.1/AV:A/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N

with a score of 6.5 (Medium).

#### C. Mapping of attacks and common vulnerabilities and exposures (CVE)

The game CATS requires players to address six attack scenarios. Each attack scenario simulates a step-by-step attack consisting of six attack actions. The attack actions are possible due to the vulnerabilities that might exist in cloud assets or are due to misconfigurations. In this work, we endeavored to map each attack action to an existing vulnerability in the first place. When there were no suitable ones to take, we derived the CVSS Vector of our own.

The CVE mapping process involved searching with keywords related to basic attacks from MITRE ATT&CK database, identifying several CVE records, and selecting the one with the highest vulnerability score to apply the worst-case scenario in the game. It's important to note that the game serves as a simulation to provide an understanding of fundamental concepts. While the mapping process is thorough, it doesn't cover all possible real-world scenarios.

For instance, brute Force is an attack technique that adversaries may use to gain access to accounts when passwords are unknown or when password hashes are obtained. Without knowing the password for an account or set of accounts, an adversary may systematically guess the password using a repetitive or iterative mechanism [17]. In our game, we mapped this attack action to CVE-2020-14494. This vulnerability is due to a flaw in an authentication mechanism that does not provide sufficient complexity to protect against brute force attacks, which may allow unauthorized users to access the system after no more than a fixed maximum number of attempts [18]. CVE-2020-14494 is a critical vulnerability, and the CVSS vector is CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H with a score of 9.8 (Critical) out of 10.0. For all the mapped vulnerabilities, we applied the CVSS version 3.1.

#### D. Mapping of attack and defense actions

Multiple defense actions can defend one attack action, and one defense action could also mitigate multiple attack actions. This many-to-many mapping is illustrated in MITRE ATT&CK cloud matrix [12]. For instance, the defense "Application Developer Guidance" refers to the defense that developers should be trained to develop applications securely. This activity is helpful in preventing the attack "Abuse credential" when the debugging and development access to the application cannot be misused.

#### E. Algorithm walk-through of an example

In this section, we walk through the algorithm with a concrete example: the attacker applies "Abuse Credential" and "Cloud Infrastructure Discovery" in step 1 to gain access. In step 2, the attacker uses the attack actions (AA) "Abuse Trusted Relationship," "Monitoring Escaping," and "Impair Defense" to launch the attack. In step 3, the attacker conducts "Resource Hijacking" to make an impact. This is shown in the right part of Figure 1.

Within one step, attack actions (AAs) are considered in parallel, which means the attack success rate for this step will only be reduced to 0% if all attacks within this step are mitigated to 0%. If there is one undefended attack with a 100% success rate, the attack success rate would be 100%. This case symbolizes that the attacker finds an open vulnerability and applies an exploit; therefore, they can bypass this step without any trouble. We assume the normalized percentage of an attack  $m$  in step  $n$  is  $p_{nm}$ , then the attack success rate of step  $n$ ,  $p_n$ , is:

$$p_n = 1 - \prod_{1 \dots m} (1 - p_{nm}).$$

The defense success rate of step  $n$  is:

$$\prod_{1 \dots m} (1 - p_{nm}).$$

Then we consider the three steps in serial order. If an attacker attacks the cloud asset in a three-step approach, he or she must succeed in all the steps to finally compromise the cloud asset. Therefore, if we assume the final attack success rate is  $p$ , then:

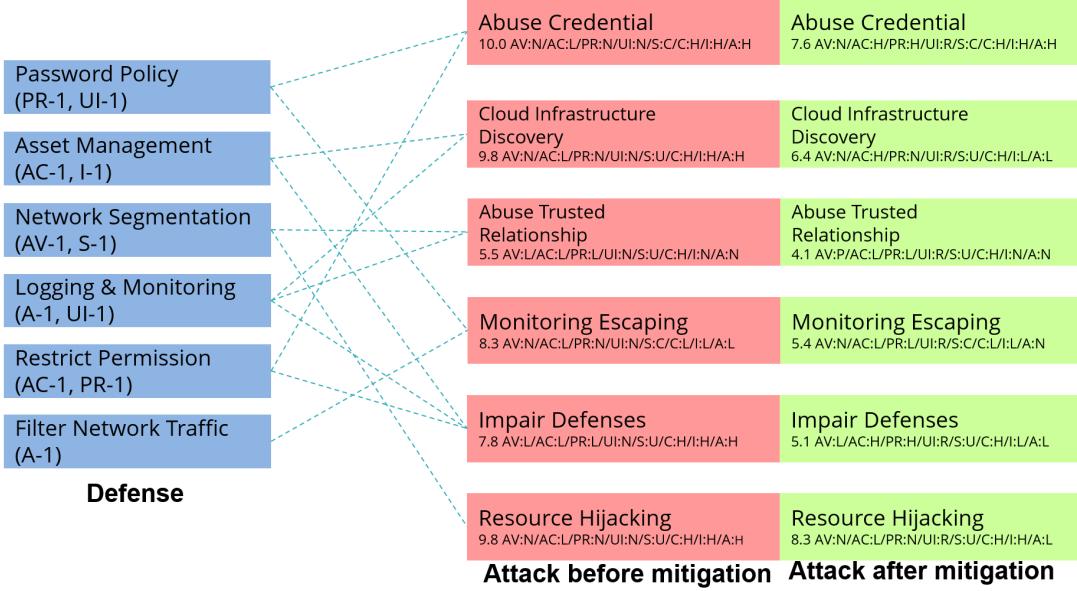
$$p = \prod_{1 \dots 3} (p_n).$$

The final defense success rate  $p_{final}$  can be derived:

$$p_{final} = 1 - p.$$

Following this logic, we can derive the final defense success rate of 3% ( $p_1 = 100\%$ ,  $p_2 = 98\%$ ,  $p_3 = 98\%$ ). The defense actions are shown in the left part in Figure 1 marked as "Defense". We assume that those cards are assigned to the correct role, therefore the impact remains valid. After applying the defense actions (DA), as shown on the right of Figure 1, the CVSS base score of each attack action (AA) drops. After applying the calculation on the new values, we

Fig. 1. Mapped vulnerability before mitigation.



derive that the defending probability has increased from 3% (before mitigation) to 34% (after mitigation,  $p_1 = 91\%$ ,  $p_2 = 86\%$ ,  $p_3 = 83\%$ ). This is still not good enough to solve the scenario, but there will be hints proposed to help the player improve the quality of the defense plan and finally achieve the required threshold.

Note that the players need to assign the activities to the correct roles in the game. If a defense card and activity are incorrectly assigned, it is ineffective in mitigating the attack and its contribution to the defense is "zero". Our walkthrough assumes that all cards are assigned to the proper roles.

## V. EVALUATION

In this section, we present the method we use to evaluate the implemented "Evaluator" algorithm. Then, we show the collected results and discuss them.

### A. Evaluation Method

We evaluate the new algorithm by applying it to in-game events and comparing the statistics of the new algorithm with the previous version of CATS. At the end of each game event, we also invited the participants to an open discussion to exchange ideas and collect feedback in the form of semi-structured interviews (SSI). We organized ten game events before refining the evaluator algorithm and two with the new Evaluator algorithm. For the two-game events in the current design iteration, 24 industrial practitioners participated as players. The game event took place on 2023-03-22 (14 players) and 2023-04-19 (10 players). In all of the 12 game events, the players are practitioners in the industry with similar experience levels. Further demographic information was not collected since it was irrelevant to the data analysis.

### B. Evaluation Results

We present in Table I the comparison of the "historic" and the new Evaluator algorithm. The evaluation results were obtained through in-game data collection and analysis.

The in-game data includes details of each player's actions, submissions, and outcomes during the gameplay. This granular data allowed us to conduct a detailed analysis of player performance, submission patterns, and defense strategies' effectiveness. The in-game data was processed and analyzed to generate the quantitative metrics presented in Table I. The data of the two-game events are combined with a total sample number of 24.

Comparing the players' performance before (sample number 123) and after the algorithm refinement (sample number 24), we observed that with the new algorithm, the players took fewer submissions per scenario to solve the scenario. The average time the players spent on each scenario drops slightly in the new algorithm. On average, the players tend to make more improvement in terms of increasing defense success rate in each submission in the new algorithm. However, in the game events with the new algorithm, players seem to make more mistakes in the role assignment per scenario.

### C. Discussion

Participants enjoyed the CATS game and found it beneficial for enhancing their cloud security knowledge. This remained unchanged for the ten historical game events and the two new game events. The use of CVSS in CATS allows for simulating attacks and reflecting more elements from reality. All metrics combined from table I, we could conclude that the algorithm refinement with more elements reflecting reality in the current design iteration did not significantly increase the game's difficulty.

TABLE I  
GAME EVENT STATISTIC: COMPARING THE IMPROVED ALGORITHM TO THE ORIGINAL ALGORITHM

	Original Algorithm	New Algorithm
Average no. of submissions per scenario	7.85	7.30
Average time to solve each scenario	5 min. 45 sec.	5 min. 18 sec.
Average improvement in each submission	5.21	6.55
No. of mistakes in role assignment per scenario	2.11	3.04

Post-game discussions through SSIs offered qualitative insights, addressing challenges and suggesting improvements. A comparison between the original and new algorithms revealed consistent participant engagement without increased difficulty, ensuring a steady level of challenge.

## VI. CONCLUSION AND FUTURE WORK

In our work, we improved the core evaluator algorithm with CVSS and MITRE ATT&CK. By involving the mentioned elements above, we managed to:

- 1) Consider the different impacts of each defense action individually.
- 2) Enhance the real-world characteristics of the simulated CATS game without significantly increasing the difficulty.

The study is conducted under the guidance of the design science paradigm, which is an approach that combines scientific research with practical design to create innovative solutions to real-world problems. As for the evaluation, we organized two game events with industry practitioners. The collected feedback shows that the players find the game enjoyable and helpful in raising awareness about the challenges in cloud security and exercising the skills for designing a successful defense strategy.

The goal of this design iteration was to enhance the realism of attack scenarios and improve the evaluator algorithm to provide players with a better understanding of the consequences of their defense strategies. The fact that players reacted similarly suggests that the difficulty level was effectively maintained, preventing an unintended increase in complexity. This outcome aligns with our objective to refine the algorithm without imposing unnecessary challenges on the players.

In our future work, we would like to conduct more game events and invite cybersecurity experts to review the mapping and the algorithm workflow. Based on the collected feedback, further improvement and adjustment can be carried out.

## ACKNOWLEDGMENT

This research task was partially supported by Fundação para a Ciência e a Tecnologia, I.P. (FCT) [ISTAR Projects: UIDB/04466/2020 and UIDP/04466/2020]. Ulrike Lechner acknowledges funding for project CONTAIN by the Bundesministerium für Bildung und Forschung (FKZ 13N16581). Tiange Zhao and Tiago Gasiba acknowledge the funding provided by the Bundesministerium für Bildung und Forschung (BMBF) for the project CONTAIN with the number 13N16585.

## REFERENCES

- [1] T. Zhao, U. Lechner, M. Pinto-Albuquerque, E. Ata, and T. Gasiba, “Cats: A serious game in industry towards stronger cloud security,” in *Ubiquitous Security*, G. Wang, K.-K. R. Choo, J. Wu, and E. Damiani, Eds., Singapore: Springer Nature Singapore, 2023, pp. 64–82.
- [2] T. Zhao, U. Lechner, M. Pinto-Albuquerque, and E. Ata, “Cloud of Assets and Threats: A Playful Method to Raise Awareness for Cloud Security in Industry,” in *Third International Computer Programming Education Conference (ICPEC 2022)*, ser. Open Access Series in Informatics (OASIcs), A. Simões and J. a. C. Silva, Eds., vol. 102. Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022, pp. 6:1–6:13. [Online]. Available: <https://drops.dagstuhl.de/opus/volltexte/2022/16610>
- [3] T. Zhao, U. Lechner, M. Pinto-Albuquerque, and D. Ongu, “An ontology-based model for evaluating cloud attack scenarios in cats – a serious game in cloud security,” in *2023 IEEE International Conference on Engineering, Technology and Innovation (ICE/ITMC)*, 2023, pp. 1–9.
- [4] N. Hänsch and Z. Benenson, “Specifying IT security awareness,” in *25th International Workshop on Database and Expert Systems Applications*, IEEE. IEEE headquarter address3 Park Avenue, 17th Floor New York, NY 10016-5997 USA: IEEE, 2014, pp. 326–330.
- [5] R. Dörner, S. Göbel, W. Effelsberg, and J. Wiemeyer, *Serious Games: Foundations, Concepts and Practice*. Springer Nature One New York Plaza, Suite 4600 New York NY 10004-1562 USA: Springer, 2016.
- [6] A. Shostack, “Tabletop security games & cards,” <https://shostack.org/games.html>, 2021.
- [7] S. Hart, A. Margheri, F. Paci, and V. Sassone, “Riskio: A serious game for cyber security awareness and education,” *Computers & Security*, vol. 95, p. 101827, 2020.
- [8] L. S. Ferro, A. Marrella, T. Catarci, F. Sapiro, A. Parenti, and M. De Santis, “AWATO: A Serious Game to Improve Cybersecurity Awareness,” in *HCI in Games*, X. Fang, Ed. Cham: Springer International Publishing, 2022, pp. 508–529.
- [9] C. C. M. W. Group, “Cloud security alliance: Cloud controls matrix v4,” <https://cloudsecurityalliance.org/artifacts/cloud-controls-matrix-v4/>, 2021.
- [10] ISO27017, “ISO/IEC 27017:2015 Information technology — Security techniques — Code of practice for information security controls based on ISO/IEC 27002 for cloud services,” <https://www.iso.org/standard/43757.html>, 2015.
- [11] ISO27018, “ISO/IEC 27018:2019Information technology — Security techniques — Code of practice for protection of personally identifiable information (PII) in public clouds acting as PII processors,” <https://www.iso.org/standard/76559.html>, 2019.
- [12] T. M. Corporation, “MITRE ATT&CK cloud matrix,” <https://attack.mitre.org/versions/v8/matrices/enterprise/cloud/>, 2020.
- [13] A. Hevner, “A three cycle view of design science research,” *Scandinavian Journal of Information Systems*, vol. 19, pp. 1–6, 2007. [Online]. Available: <http://aisel.aisnet.org/sjis/vol19/iss2/4>
- [14] N. Organization, “Vulnerability metrics,” 2023. [Online]. Available: <https://nvd.nist.gov/vuln-metrics/cvss>
- [15] MITRE, “Network Service Discovery,” <https://attack.mitre.org/versions/v13/techniques/T1046/>, 2017.
- [16] ———, “CVE-2020-1206 Detail,” <https://nvd.nist.gov/vuln/detail/cve-2020-1206>, 2020.
- [17] ———, “Brute Force,” <https://attack.mitre.org/versions/v13/techniques/T1110/>, 2017.
- [18] ICS-CERT, “CVE-2020-14494 Detail,” <https://nvd.nist.gov/vuln/detail/CVE-2020-14494>, 2020.



# **Posters**



# Using Generative Artificial Intelligence Tools in Software Engineering Courses

Soma Datta

Department of Engineering

University of Houston- Clear Lake

Houston, USA

0000-0003-3701-6957

This pilot study focuses on allowing students to use generative Artificial intelligence (AI) tools for their learning and assignments. Therefore, the study looks to improve the assignments to assess their learning. Students in their course have both formative and summative assessments. Both these assessments consist of writing, quizzes, and presentations. These students are both from the undergraduate and graduate levels. The study seeks to make the assessments sustainable for all teaching levels. The change in writing assessment would help to assess students better. The assignments are tested on ChatGPT and Bard to check if a student gets a passing grade using an AI tool.

The research question for this study is: do instructors need to redesign their assessments with the dawn of widely available AI tools?

The data set used in this study was 39 undergraduate students and 30 graduate students. These students were from both the Fall and Spring semesters. The maximum class size was 19 students. The pilot study was done on software engineering in the course requirements engineering, into software engineering, and agile software development courses.

As evaluated in [1], this study focuses on software engineering courses. The formative assignments are typical questions from the textbook that are peer-reviewed. These assessments help them to read their peer's viewpoints. It is a requirement for course completion. They also get graded by their peers. Assignments graded by their peers are not added to their final grade. Since these are directly from textbooks, the AI tools [2] do not affect them.

In the past, the summative assignments [3] were writing and projects. In this study, these writing assessments are reviewed to be tested for learning. Students are allowed to use any AI tool for their writing assessments. Hence, it is difficult to identify if the student has learned the given topic. However, they cite any AI material used as they did earlier for other references. These assessments need to be modified.

They have several writing assessments; students present some assignments in in-person classes. The challenge the instructor faces is the extra time needed to have them present. Class presentation activity would be challenging to implement in a large class.

Before the dawn of these AI tools that are freely available, such as ChatGPT/Bard, one of the writing assessments still works well. That is the lighting talk assessment. In this assessment, students are given a topic ahead of time, and they prepare 15 slides along with a five-minute presentation. They present this in class while having the slides automatically scroll. This has proved to be a helpful learning assessment because they have to practice well before their presentation. This assessment has still proved to work well in these two

semesters for in-person classes. Hence, no changes were required.

For another assignment, as a pilot study, students prepare lectures for the given slides. The feedback from students was as follows. They wanted the instructor to lecture instead of listen to their peers. They said it was too much work and would instead prepare for the class and do a voice-over of the given slides. While discussing with students, they would be ready to take quizzes or knowledge check questions in class during lecture time.

A short answer type assessment with multiple choices, true/false, and fill-in-the-blanks for each chapter helped better for assessment. Students were asked to be prepared to take the quiz during class time. This assessment worked well in assessing if the student came prepared to class and understood the topic discussed. Various writing assignments were converted to short-type, quick class assessments.

In a different in-class writing assessment, students had to be creative when asked to present their writing assessment in five minutes or shorter. These assessments also had a question-answer session. Hence, students had to understand the concept. This assignment still works well. Thus, it can safely be concluded that in-class assessment can continue with little supervision.

Another writing assessment was a two-page writing on a given topic. Most students did well. Most students in these semesters have used AI tools. After checking the topic using AI tools, it was found that it gives students a good grade. Hence, this assessment needs to be revised to evaluate student learning better. A writing assessment integrated with a presentation or Q&A in class could still be safe for assessing students' learning.

In another course in which project-based learning [4] was the fundamental pedagogy; students use various open sources to develop their projects. Students create their project component-based and use APIs to build the project. They have added machine learning to their software product. Some of the tools they have started using are ChatDev IDE and Devika. Project-based learning still works well with the availability of AI tools.

In the future, the study will test the assessments for online students. Online students might not be available at any time convenient to the instructor. Hence, in the future, this study will find ways to add online students.

**Keywords— Generative Artificial Intelligence, Software Engineering, Assessment, AI tools**

## Reference

1. Nikolic S, Daniel S, Haque R, et al (2023) ChatGPT versus engineering education assessment: a multidisciplinary and multi-institutional benchmarking and analysis of this generative artificial intelligence tool to investigate assessment integrity. *European Journal of Engineering Education* 48:559–614.  
<https://doi.org/10.1080/03043797.2023.2213169>
2. Bull C, Kharrufa A (2023) Generative AI Assistants in Software Development Education: A vision for integrating Generative AI into educational practice, not instinctively defending against it. *arXiv.org*.  
<https://doi.org/10.48550/arxiv.2303.13936>
3. Burke K (2014) *Balanced Assessment: From Formative to Summative*. Solution Tree, Bloomington, UNITED STATES
4. Helle L, Tynjala P, Olkinuora E (2006) Project-based learning in post-secondary education--theory, practice and rubber sling shots. *Higher Education* 51:287–315

# You & AI: a research computing hackathon

## Poster Abstract

James DesLauriers  
Research Computing & Data Science  
Graduate School  
Imperial College London  
London, England  
j.des@imperial.ac.uk

Liam Gao  
Research Computing & Data Science  
Graduate School  
Imperial College London  
London, England

Katerina Michalickova  
Research Computing & Data Science  
Graduate School  
Imperial College London  
London, England

Chris Cooling  
Research Computing & Data Science  
Graduate School  
Imperial College London  
London, England

John Pinney  
Research Computing & Data Science  
Graduate School  
Imperial College London  
London, England

**Abstract**—In June 2023, the Imperial College London Graduate School’s Research Computing and Data Science group invited thirty PhD students for a one-day Hackathon on AI-assisted programming. This poster abstract presents our experiences in planning and running the event, and shares student and organiser reflections on the event and where it might lead.

**Keywords**—generative ai, training, ai-assisted programming

### I. INTRODUCTION

The impact of generative AI has a far reach. It is currently disrupting and is expected to further disrupt many industries. Academia is one industry where that disruption is expected to be greatest, both in the areas of teaching and learning, and in research [1]. Generative AI in these areas independently has been the focus of several papers. Less prevalent in the literature is work at the intersection of these two areas, where doctoral candidates and researchers learn to use and leverage generative AI for their research. One unexplored area in particular is researcher training in AI-assisted programming, to support the academic research that depends on computation, data science, and adjacent technologies.

The Research Computing and Data Science (RCDS) Group at the Imperial College London Graduate School provides programming skills training to doctoral candidates to support their research. Generative AI would undoubtedly create many new opportunities for learning and teaching, but of immediate interest to the group was how researchers with already established programming experience could start to leverage generative AI to support their work.

With this idea in mind, RCDS began preparations for a workshop that would introduce attendees to generative AI, prepare them with a development environment for AI-assisted programming, and challenge them with a novel computational research problem.

### II. PREPARATION

The first thought of a workshop for AI-assisted programming came in April 2023, roughly a month after Open AI’s GPT4 was released and when the world was seeing just what generative AI could do. At that stage, very little had been published in AI-assisted programming training, and none of our peers had yet run any sort of training in the space. As such, the organisation of the workshop was more organic than systematic. We reached out to contacts at GitHub and

Microsoft to find speakers who could introduce generative AI, provide access to appropriate tools, and support us with training on the day. Microsoft agreed to send a speaker, and GitHub offered a team of GitHub Campus Experts and temporary GitHub Copilot subscriptions for all attendees.

For the event format, we decided on a mixed workshop-hackathon, with attendees placed into small interdisciplinary groups after the morning talks. Hackathons have a variety of applications, and are appreciated in academic settings as drivers of both innovation and learning [2] [3]. In particular, hackathons with an interdisciplinary audience that focus on data analysis projects have seen high levels of knowledge sharing and learning take place [4].

With the event format determined, an RCDS team member with expertise in nuclear engineering crafted a three-part data analysis research problem that closely mimicked real research. Within each part, there were several parallel tasks that could be completed by individuals or sub-teams within each group. For example, in part one, there were retrieval, cleaning, and analysis tasks that did not depend on each other. The parts of the problem increased in complexity, with part three being the most challenging. The full problem is available<sup>1</sup> on GitHub and includes a sample solution of roughly 600 lines of code. Nuclear engineering was chosen as the problem domain because of the expertise within RCDS, and because for most attendees, this would be a novel and hopefully challenging problem.

With the technical aspects of workshop planning done, the team settled on *You & AI* as a name for the event, booked a room on campus, arranged catering for lunch and coffee breaks, and began advertising. Thirty spaces were quickly booked by PhD students across various departments at Imperial, and the event was ready to go.

### III. THE EVENT

The workshop ran at the end of June 2023, and the day was organised into two parts: a morning session with talks and demonstrations from Microsoft and the GitHub Campus Experts, and an afternoon session where students worked in teams to code up the solution to the research problem.

#### A. Morning Workshop

Since attendees were all competent programmers, the aim here was to introduce the topic of generative AI and demonstrate an approach to generating code with a large

<sup>1</sup> <https://github.com/cmcooling/Decay-Chains>

language model (LLM). The first speaker, from Microsoft, gave a high-level overview of the current landscape of generative AI within their organisation. Next, a GitHub Campus Expert gave a live demonstration of GitHub Copilot, coding up the game of pong on-the-fly using JavaScript and HTML, by interacting with the GitHub Copilot chatbot.

### B. Afternoon Hackathon

Students were split into groups and were introduced to the nuclear engineering problem by the RCDS expert. GitHub Campus Experts registered all attendees with temporary Copilot subscriptions and supported them with enabling Copilot in their VS Code development environments. Students then worked in teams, using generative AI to design and implement a solution to the problem. Because of the focus on GitHub Copilot earlier in the day, it was favoured by students.

## IV. OUTCOMES AND LESSONS LEARNED

All teams made significant progress towards a solution and completed the first two parts. Due to time constraints, no team completed all the activities in part three.

At the end of the day, we asked students to reflect on their experiences in AI-assisted programming. Of those in attendance, 17 attendees responded to the survey. Of particular interest were the intentions of attendees to engage with generative AI for programming related tasks before and after they had attended the hackathon, with all students reporting they would be more likely to use generative AI for programming in the future. This is visualized in Fig. 1.

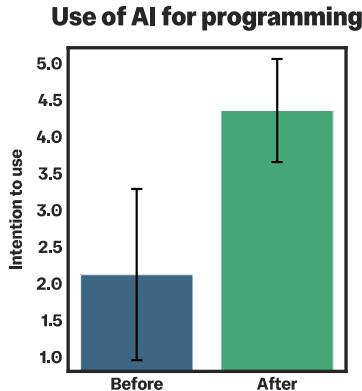


Fig. 1. Averaged responses to the end-of-day survey question: "How much did/will you use AI-Assisted Programming?" (N=17).

There were mixed opinions on the utility of the responses generated, perhaps because interactions with an LLM are dependent on user prompts. We determined that future events should spend even more instruction time on prompting the AI and iterating with it towards the best response, with one student even saying that "more specific prompting techniques would have been good". Given that our attendees were all competent programmers, we asked for their reflections on the quality of the code produced, across four criteria. This data is visualized in Fig. 2.

We captured student sentiment about generative AI, with students calling it "impressive" and "more advanced than thought" and noting it was particularly useful for "Q/A and documentation" and simple tasks. At least two other students

found that it made "debugging hard" and "needs steering" or "pre-cursor knowledge". With regard to the event, one student said it was a "very good format to demonstrate use case and then have hands on learning" and another that it was "well structured with [a] good practical activity".

### Student evaluation of AI code quality

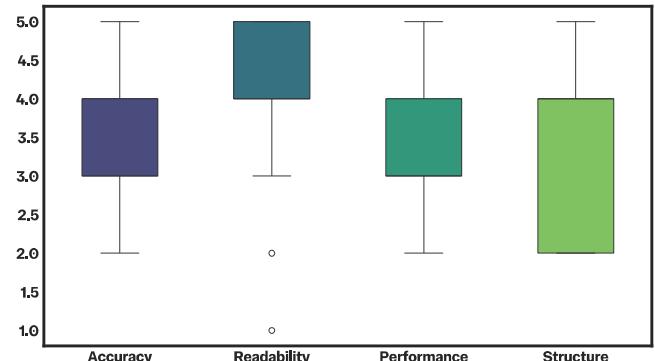


Fig. 2. Student ratings (1: Poor to 5: Excellent) of different criteria of code quality, captured by the end-of-day survey question: "How would you rate the quality of the code generated by AI today?" (N=17).

## V. FUTURE WORK

Our objectives with the You & AI event were varied. We wanted to engage with training in AI-assisted programming and understand if a mixed workshop-hackathon with a large research problem would be an appropriate format for learning. We wanted to gauge whether and how our PhD students were engaging with generative AI for their research and demonstrate by experience how that engagement might look. We also wanted to understand what students thought about the quality of the code being generated by AI. Another hackathon is planned for summer 2025, with the theme of optimization.

We are drawing on all this experience to develop teaching materials in AI-assisted programming to further support PhD students at Imperial, with a pilot course planned for next academic year. Among our considerations for the course are the importance of testing, validation and verification when working with LLMs, and what base understanding of programming is needed before engaging with generative AI.

## ACKNOWLEDGMENT

We thank Microsoft and GitHub for their support of our hackathon, and would like to extend special thanks to the GitHub Campus Experts who supported us on the day.

## REFERENCES

- [1] Dwivedi, Yogesh K., et al. "'So what if ChatGPT wrote it?' Multidisciplinary perspectives on opportunities, challenges and implications of generative conversational AI for research, practice and policy." International Journal of Information Management 71 (2023): 102642.
- [2] J Groen, Derek, and Ben Calderhead. "Science hackathons for developing interdisciplinary research and collaborations." Elife 4 (2015): e09944.
- [3] Pe-Than, Ei Pa Pa, and James D. Herbsleb. "Understanding hackathons for science: Collaboration, affordances, and outcomes." Information in Contemporary Society: 14th International Conference, iConference 2019, Washington, DC, USA, March 31–April 3, 2019, Proceedings 14. Springer International Publishing, 2019.
- [4] Ghouila, Amel, et al. "Hackathons as a means of accelerating scientific discoveries and knowledge transfer." Genome research 28.5 (2018): 759–765.

# MASS, Marburg university auto ASess System

Steffen Dick    Christoph Bockisch

*Philipps-Universität Marburg, Germany*

35032 Marburg, Germany

{dickst,bockisch}@mathematik.uni-marburg.de

**Abstract**—In computer science, in particular in early programming courses, students often do not fully solve an assignment on their first try. While teachers and tutors can provide individual feedback in such cases, they are only available to a limit. Automatic feedback tools can be available 24/7 but are limited in how individual their feedback is. The MASS system is an automatic feedback tool that can flexibly be configured by teachers on a per-assignment basis to provide individual feedback tailored to their courses and the current learning stage of students.

**Index Terms**—education, digital, testing, feedback, automation

## I. PROBLEM STATEMENT & RELATED WORK

Early in their university education, students of Computer Science are supposed to complete a course in programming. Within these courses, they usually learn the basic constructs of a given programming language and how to use them to solve problems throughout the semester. Providing timely and constructive feedback is difficult for lecturers and tutors, especially in courses with high numbers of students. Automatic feedback can help lecturers to stem the tide of feedback required by students. In contrast to human feedback, automatic feedback is available 24/7, has a very low response time and is available repeatedly. Furthermore, Cavalcanti et al. found improvement in student performance when automatic feedback was introduced [2]. Several tools that provide automatic feedback to students already exist. However, these tools are often difficult to use and their feedback is hard to grasp for students in the beginning of their studies. Also, most of these tools base their generated feedback on the use of unit tests, for example, JACK [5] and INLOOP [6].

## II. SOLUTION APPROACH

Because of this, we have developed MASS, the Marburg University auto-assessment system, which consists of several existing tools packaged into one easily configurable software. Thereby, MASS currently focuses on providing feedback on Java programming assignments, and many of the features presented here are only implemented for Java. The overall approach, however, is language agnostic and we are working on support for other programming languages. The different components, called *checkers*, analyse different aspects of code submitted by students. MASS has general rules, where teachers can configure the systems' strictness, adapt rules according to their individual requirements or to the

current knowledge of students in their courses and configure messages more individually on a per-assignment basis. MASS provides an accessible, 24/7 available tool set that is highly configurable to give understandable feedback for students in various stages of learning and various courses. The full source code of MASS is available at our open-source Git repository <https://github.com/qped-eu/MASS-checker>.

## III. IMPLEMENTATION

MASS is designed as a framework for generating feedback for programming assignments. It is controlled in a universal way, simply by providing a configuration file. MASS can then be started from the command line, which makes it easy to integrate into different e-learning environments. The configuration is provided by the lecturer per assignment and specifies the assessment criteria that should be applied, as well as rules for providing suitable feedback. The e-learning environment through which students submit their solution merges the solution with into the teacher's configuration, places the result in a JSON formatted file of a predefined name and runs MASS from the command line. After execution, MASS will add the generated feedback as a MarkDown string to the JSON file. The e-learning environment can read the result from the file and present the feedback to students.

We rely on an e-learning environment for a submission facility and to execute MASS in a safe way. Since MASS generally executes the student solution, isolating the execution is important to prevent malicious code, either submitted by accident or on purpose. For example, the Quarterfall Cloudcheck server<sup>1</sup> can be used for submissions, and MASS has actually been developed to be used with this Cloudcheck server. Due to the generality of the way of interaction, it can, nevertheless, be used with other e-learning environments.

When MASS is executed, it inspects the provided configuration and determines the origins of source code required for the feedback generation. MASS then combines the source code and compiles it. Next, from the configuration, it determines which checkers should be applied to the submitted solution. Each checker returns a list of feedback items that are supposed to be presented. A feedback item is structured and usually contains the following information: A description, a reference into the teaching material where the corresponding concept is covered, an example of the problem type and how to fix it, the location where the problem was found. Lastly, MASS collects

This work is partly funded by the Erasmus+ project *Quality-focused Programming Education (QPED)*, 2020-1-NL01-KA203-064626.

<sup>1</sup>The Quarterfall homepage: <https://www.quarterfall.com>

the feedback items from used Checkers and combines them into a String, which is then added back into the file.

*Syntax Checker:* Source code can be provided by students either as a ZIP file or as a text. The ZIP file is provided as a URL from which MASS can download the file and extract it to a dedicated source folder. If code is provided as text, it may contain either a full Java compilation unit or only code snippets. MASS uses the Eclipse parser to determine what is actually provided in the answer and extends it by wrapping instructions in a default method or class, which is also written to the same folder. Secondly, the teacher's code is always provided as a URL, which MASS downloads and extracts to the dedicated source folder. The teacher's code is always extracted last and potentially overwrites code, which is intentional because teachers configure the feedback tool, assuming that their own code is used. To avoid overwriting student code, file names should be specified within the assignment.

Next, the source code is compiled. Compiler errors in the teachers' code are reported with a note to ask for help: since students can not access the teacher code, they have no chance of fixing the problem. *Syntax Checker*, which is the only mandatory checker in MASS, internally uses the Java Compiler [1] and repackages the output into easily understandable feedback. In case of compiler errors, MASS stops execution and reports the feedback items.

The syntax checker can be configured by choosing one of three different levels: beginner, intermediate and advanced. They differ in how much explanation of the type of error they contain. The beginner level, obviously, is more verbose and includes an example of how to solve the type of error. The advanced level provides the most succinct message and basically corresponds to the compiler's error message.

If no compiler errors are found, MASS assembles all relevant source files in one folder and compiles them into the same folder. Next, MASS inspects the remaining configuration file to determine, which further checkers to execute.

*Style-Checker, Solution-Approach-Checker:* We use PMD to analyze the submitted code regarding violations of styling guidelines. Teachers can configure naming conventions for this check in terms of regular expressions. Furthermore, MASS provides tailored feedback items including elaborate descriptions and examples. Teachers can also choose from the same three proficiency levels for the displayed feedback as with the syntax checker, which will also influence how comprehensive the set of applied style checking rules is. The solution approach checker slightly extends the style checker to assess the language constructs that a student used to solve a problem. It can, for example, detect forbidden use of recursion or limit the number and kind of loops used within the solution.

*Class-Info-Checker, Metrics-Checker:* These two checkers apply metrics to the source code to assess the class design. The first checker can provide feedback on different access modifiers or the inheritance declarations. Teachers can configure rules for the expected modifiers or expected super classes per field, method or class. The metrics checker applies the standard Chidamber & Kemerer metrics suite [3]

(e.g., *coupling between objects, lack of cohesion or depth of inheritance tree*). Teachers can configure thresholds for each metric and provide custom feedback messages.

*Test-Coverage-Checker:* This checker performs a dynamic analysis, i.e., it executes all unit tests found in the code and determines which code is executed (or covered) during the tests. Teachers can, among others, include tests with assignment specific assertions and failure messages, which is also supported by other automatic feedback tools. What is more, with the MASS test coverage checker, they can provide a private sample solution with the teacher's code and let the students' test run against that sample solution; code lines within the sample solution that are not covered during the test run indicate missing test cases and teachers can configure appropriate feedback messages.

#### IV. RESULTS & FUTURE WORK

MASS has been used in a first semester *object-oriented programming* lecture at the university of Marburg where we gathered feedback from students and improved MASS based upon their criticism. Firstly, we improved the efficiency of MASS and cut down on the execution time. Secondly, we plan to support other programming languages than Java. Thirdly, we will extend the framework to allow generally configuring feedback based on further conditions. For example, become more directive the more often a student attempts and fails to submit a solution. Lastly, we are actively developing a plugin for Eclipse and IntelliJ that allows getting feedback within the IDE instead of having to use an external website. Furthermore, we published our findings in [4] where we found that including MASS has had a positive effect on our students' comprehending of software quality in both, their subjective view and in a more objective view.

#### REFERENCES

- [1] Edward Aftandilian et al. "Building useful program analysis tools using an extensible Java compiler". In: *2012 IEEE 12th International Working Conference on Source Code Analysis and Manipulation*. IEEE. 2012.
- [2] Anderson Pinheiro Cavalcanti et al. "Automatic feedback in online learning environments: A systematic literature review". In: *Computers and Education: Artificial Intelligence* 2 (2021).
- [3] Laila Cheikhi et al. "Chidamber and Kemerer Object-Oriented Measures: Analysis of their Design from the Metrology Perspective". In: *International Journal of Software Engineering and Its Applications* 8 (Mar. 2014).
- [4] Steffen Dick, Teresa Dreyer, and Christoph Bockisch. *TILE and MASS, a retrospective*. Software Engineering im Unterricht der Hochschulen (SEUH 2024). in press.
- [5] Michael Goedicke, Michael Striewe, and Moritz Balz. *Computer aided assessments and programming exercises with JACK*. Tech. rep. ICB-Research Report, 2008.
- [6] Martin Morgenstern and Birgit Demuth. "Continuous Publishing of Online Programming Assignments with INLOOP." In: *Software Engineering (Workshops)*. 2018.

# FeedbackPulse: GPT-Enabled Feedback Assistant for Software Engineering Educators

1<sup>st</sup> Yiwen Liao

*University of New South Wales  
Sydney, Australia  
yiwen.liao@student.unsw.edu.au*

2<sup>nd</sup> Yuchao Jiang

*University of New South Wales  
Sydney, Australia  
yuchao.jiang@unsw.edu.au*

3<sup>rd</sup> Zhangpeng Chen

*University of New South Wales  
Sydney, Australia  
zhangpeng.chen@student.unsw.edu.au*

4<sup>th</sup> Basem Suleiman

*University of New South Wales  
Sydney, Australia  
b.suleiman@unsw.edu.au*

**Abstract**—In response to the growing enrolment in software engineering programs, there is a pressing need for scalable methods to provide effective feedback for students. We designed a GPT-driven interactive assistant, FeedbackPulse, to aid educators in delivering high-quality feedback and alleviating their workload. FeedbackPulse provides real-time, personalised feedback suggestions to educators for improving their feedback to students. Preliminary evaluations of FeedbackPulse in a software engineering course have demonstrated its promising capabilities.

## I. INTRODUCTION

Feedback is essential for students to improve their work and learn new skills. However, generating effective feedback on students' work is time-consuming and requires training, especially for new educators (e.g., tutors) [1]. Furthermore, with the increasing number of students enrolled in software engineering degrees, educators struggle to provide feedback of good quality at a large scale. Interactive guidance for educators on providing good quality feedback can ease the process. Currently, there is a lack of an automated tool that can provide educators with this interactive guidance on improving their feedback quality.

To address the challenges, we designed a feedback assistance tool named *FeedbackPulse*. *FeedbackPulse* automatically detects feedback that needs improvement in real-time and provides personalised guidance for educators to improve the feedback they give to students. *FeedbackPulse* is published as a Google Chrome extension and is available for download<sup>1</sup>.

*FeedbackPulse*'s guidance is based on the four attributes of effective feedback: specific, actionable, justified, and with proper tone. We adopted these criteria from the attributes of good feedback on creative designs [2], [3], academic work [4] and the theory of formative assessment [5]. Feedback is actionable if it guides students in improving their work, justified if it contains an explanation or reason for feedback, and specific if it is related directly to a particular part of the work rather than being vaguely referent. Moreover, maintaining a respectful and constructive tone when giving feedback fosters a positive environment.

FeedbackPulse delves into the potential of Artificial intelligence (AI) in scaling feedback on students' work. More specifically, OpenAI's GPT models emerged as a compelling choice for FeedbackPulse due to their remarkable scale and state-of-the-art performance in natural language tasks.

## II. FEEDBACKPULSE

FeedbackPulse provides interactive guidance for educators providing feedback to students, which integrates smoothly into educators' routines.

### A. WORKFLOW

FeedbackPulse utilises the GPT-3.5 Generative AI model<sup>2</sup>. Figure 1 shows an overview of the FeedbackPulse workflow.

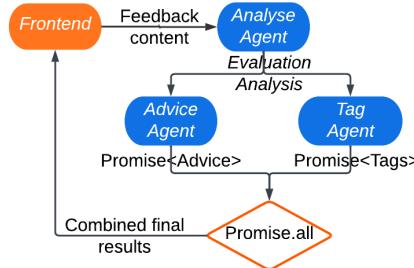


Fig. 1. FeedbackPulse Backend Workflow

1) **A Multi-Agent Approach:** As illustrated in Figure 1, FeedbackPulse deployed a multi-agent approach to address the limitations associated with single AI agents when handling complex tasks. The multi-agent approach is inspired by [6], in which ChatGPT demonstrated its potential in medical evaluations but exhibited limitations with complex or vague instructions.

Accordingly, we divided the feedback analysis task among three distinct GPT agents: Analyse Agent, Advice Agent and Tag Agent. This division allows each agent to specialise in its task, collectively ensuring more accurate and nuanced results.

**Analyse Agent:** This agent takes the feedback data and evaluates the feedback quality according to the four attributes

<sup>2</sup>OpenAI. (n.d.). GPT-3.5 Models Documentation. Retrieved October 13, 2023, from <https://platform.openai.com/docs/models/gpt-3-5>

<sup>1</sup><https://shorturl.at/aDIQV>

of good-quality feedback. For example, given a tutor's feedback that "this should be multi-line/split up into multiple lines as if it were in .sql file", Analyse Agent's evaluation about its justification is that "The tutor does not provide any justification for why the code should be split into multiple lines".

**Advice Agent:** Based on the evaluations from the Analyse Agent, the Advice Agent provides recommendations on improving the feedback. For example, in response to the Analyse Agent's input on justification, the output is "Provide a more explicit justification for why the code should be split into multiple lines".

**Tag Agent:** This agent takes the analysis from the Analyse Agent and transforms the evaluations into tags. It indicates 'thumbs up' or 'thumbs down' for actionability, justification, and specificity. It assigns a 'positive', 'neutral', or 'negative' tag for tone. For example, based on the input from the Analyse Agent, the Tag Agent outputs "Justification: thumbs down", serving as a quick reference in our interface to help users decide which attribute to examine and review the advice.

2) **AI PROMPTING:** AI prompting [7] plays an essential role in FeedbackPulse. We crafted the prompts for each of the three GPT agents, ensuring that each one focuses on a specific role during the analysis, such as giving advice, or tagging.

**Baseline dataset:** We collected feedback that educators provided to students in software engineering courses. The authors of this paper tagged the feedback according to the four feedback quality attributes. In cases where there were disparities in the tagging results among the evaluators, we established a consensus-based approach for determining the final tags.

**Prompting Engineering and Refinement Process:** We iteratively refined the prompts, comparing FeedbackPulse outcomes with a baseline dataset to tailor each GPT agent's prompt design. This process aimed to align the system's guidance with educators' insights by understanding how GPT evaluates feedback and adjusting the prompts accordingly.

## B. USER INTERFACE

FeedbackPulse displays an icon next to a selected text segment; clicking this icon opens a popup and initiates feedback analysis, as shown in Figure 2. Clicking an attribute takes the user to an advice interface for feedback refinement, as depicted in Figure 3.

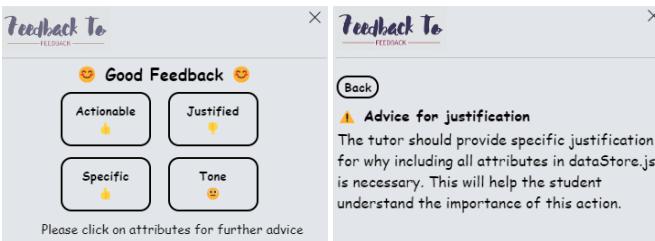


Fig. 2. Main Interface

Fig. 3. Advice Interface

## III. EVALUATION

We evaluated FeedbackPulse's effectiveness in enhancing educators' feedback on student's work by measuring its

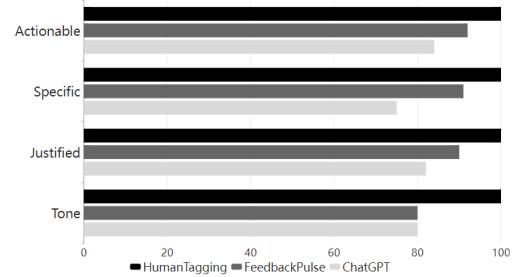


Fig. 4. FeedbackPulse Performance Evaluation

performance in advising on four key aspects: actionability, justification, specificity, and tone.

**Method:** To accomplish this goal, we collected diverse feedback samples that tutors gave to students in two software engineering courses. This evaluation set is separate from the one used for prompt refinement. The authors of this paper independently tagged these feedback samples according to the four feedback quality attributes.

We compared the evaluations from FeedbackPulse with those from the authors' tagging and ChatGPT-3.5, and calculated the accuracy in each dimension. The performance evaluation is summarised in Figure 4.

**Result:** Figure 4 demonstrates that FeedbackPulse outperforms the ChatGPT-3.5 model and almost aligns with the baseline in all four dimensions of feedback quality attributes.

However, FeedbackPulse had limitations in providing detailed feedback on highly technical subjects, due to the lack of context of the courses and assessments. For future work, plans include embedding it into educational systems for relevant feedback and simplifying the evaluation process, with future enhancements aimed at context-aware functionality to address these limitations.

## REFERENCES

- [1] Y. Jiang, B. Benatallah, and M. Baez, "Understanding how early-stage researchers leverage socio-technical affordances for distributed research support," *Information and Software Technology*, p. 107340, 2023.
- [2] T. J. Nguon, C. A. Fraser, A. S. Weingarten, M. Dontcheva, and S. Klemmer, "Interactive guidance techniques for improving creative feedback," in *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems, CHI '18*, (New York, NY, USA), pp. 55:1–55:11, ACM, 2018.
- [3] A. Yuan, K. Luther, M. Krause, S. I. Vennix, S. P. Dow, and B. Hartmann, "Almost an expert: The effects of rubrics and expertise on perceived value of crowdsourced design critiques," in *Proceedings of the 19th ACM Conference on Computer-Supported Cooperative Work and Social Computing, CSCW '16*, (New York, NY, USA), pp. 1005–1017, Association for Computing Machinery, 2016.
- [4] Y. Jiang, B. Benatallah, and M. Baez, "Rsourcer: Scaling feedback on research drafts," in *International Conference on Advanced Information Systems Engineering*, pp. 61–68, Springer, 2023.
- [5] D. R. Sadler, "Formative assessment and the design of instructional systems," *Instructional science*, vol. 18, no. 2, pp. 119–144, 1989.
- [6] T. C. Chen, E. Kaminski, L. Koduri, A. Singer, J. Singer, M. Couldwell, J. Delashaw, A. Dumont, and A. Wang, "Chat gpt as a neuro-score calculator: Analysis of a large language model's performance on various neurological exam grading scales," *World Neurosurgery*, 2023.
- [7] J. Oppenlaender, R. Linder, and J. Silvennoinen, "Prompting ai art: An investigation into the creative skill of prompt engineering," *arXiv preprint arXiv:2303.13534*, 2023.

# Presenting Experiential Educational Machine Learning Labs

Yang Liu\*, Xiaofan Que†, Dingrong Wang‡, Samuel A. Malachowsky§ and Daniel E. Krutz¶

Golisano College of Computing and Information Sciences

Rochester Institute of Technology

Rochester, NY, USA

Email: \* [yl4070@rit.edu](mailto:yl4070@rit.edu), † [xq5054@rit.edu](mailto:xq5054@rit.edu), ‡ [dw7445@rit.edu](mailto:dw7445@rit.edu), § [samvse@rit.edu](mailto:samvse@rit.edu), ¶ [dxkvse@rit.edu](mailto:dxkvse@rit.edu)

**Abstract**—Artificial intelligence (AI) is becoming increasingly prevalent in our society, leading to a growing demand for a skilled workforce in AI. Unfortunately, this demand remains unmet, particularly among smaller institutions and those serving underrepresented groups, due to resource limitations. This initiative offers two hands-on educational activities in Artificial Intelligence and Machine Learning to facilitate the integration of AI/ML concepts into foundational computing and non-computing courses. While primarily aimed at undergraduate students, the materials created can also benefit high school (grades 9-12) and graduate students in various educational settings, including traditional classrooms and outreach or after-school programs. Labs are accessible through a web browser, making them easy to adopt at all institutions, especially those with limited resources. The self-contained and hosted nature of the labs ensures that they can be adopted by institutions facing resource constraints. The complete project material is publicly available on the project website: <https://all.rit.edu>

**Index Terms**—Empathy Building Education, Experiential Education, Computing Education

## I. INTRODUCTION

We have developed a series of experiential educational labs focused on advancing AI education. Each *Accessible Learning Lab* (ALL) [2] is equipped with comprehensive learning objectives and materials, designed for seamless integration into classroom settings, and hosted online to facilitate the transition to our web-based educational platform. These labs are designed to teach fundamental AI/ML concepts to individuals, regardless of their programming background, emphasizing the broader societal implications beyond technical aspects. They offer accessible and engaging experiential learning opportunities that allow participants to understand how AI/ML influences various real-world domains. By combining theoretical knowledge with hands-on applications, our goal is to empower people to navigate the AI/ML landscape confidently. Our approach includes interactive learning modules and practical experiences that highlight the multifaceted impact of AI. We present two AI/ML-focused labs available on our ALL platform, aimed at facilitating the seamless integration of AI/ML topics into diverse computing and non-computing curricula. These labs cover the following: I) Fundamental Ethics of AI, and II) AI in Cybersecurity.

The *Accessible Learning Labs (ALL)* [2] project provides an impactful educational experience with experiential labs that are based on a variety of computing topics. Their basic

workflow is illustrated in Figure 1. The labs have demonstrated both their technical and educational effectiveness in several previous publications [1, 3, 4, 5, 6, 7, 8].

## II. DEVELOPED LABS

### A. Lab #1: Foundational Ethics of AI

The use of AI has become widespread in career and hiring practices, often screening candidates before their resumes reach a hiring manager's desk. Although this approach efficiently filters out candidates who do not meet specific company criteria, AI-based hiring and screening systems can pose significant risks and challenges. Self-learning AI systems used for candidate screening can develop biases over time, leading to the rejection of potentially suitable candidates who meet the firm's interests but are eliminated due to ingrained biases within the AI. Figure 2 illustrates the flowchart of Lab #2.

### B. Lab #2: AI in Cybersecurity

The field of AI is expanding within the realm of cybersecurity, driven by the increasing complexity of threats faced by cybersecurity experts. The integration of AI into this sector aims to improve the efficiency of cybersecurity procedures, providing engineers with streamlined processes and real-time insight into system security and potential risks. However, it is crucial to recognize that AI-powered cybersecurity presents challenges. Although automation is valuable, human intervention remains essential in cybersecurity, particularly for critical decision making. These decisions can have significant consequences, ranging from substantial financial losses to the compromise of thousands of individuals' identities. Figure 3 illustrates the flowchart of Lab #2.

### Acknowledgements

This material is based upon work supported by the NSF under grant #1825023, #2145010, and #2111152.

## REFERENCES

- [1] S. Khan, H. Moses, S. Malachowsky, and D. Krutz. Experiential learning in undergraduate accessibility education: Instructor observations. *Journal of Computing Sciences in Colleges*, 38(8):54–68, 2023.
- [2] A. L. Labs. Accessible learning labs. <https://all.rit.edu>.

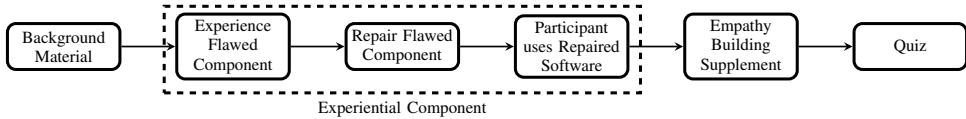


Fig. 1: Experiential, empathy-building intervention format utilized in *Accessible Learning Labs* project [2].

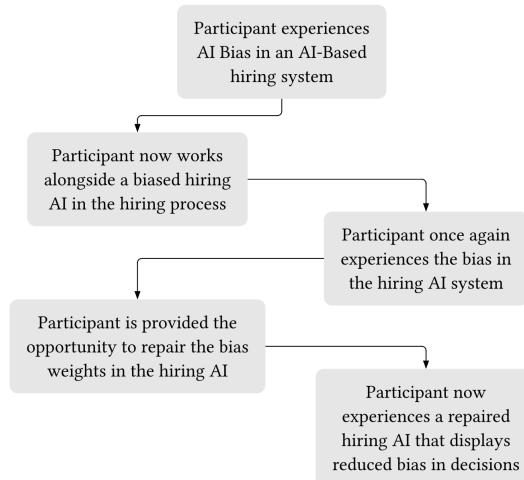


Fig. 2: The experiential, empathy-building exercise in Lab #1: Fundamental Ethics of AI demonstrated in this flowchart.

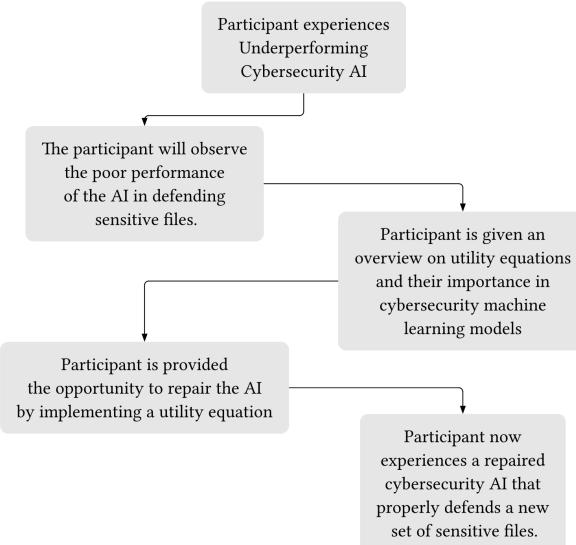


Fig. 3: The experiential, empathy-building exercise in Lab #2: AI in Cybersecurity is broken down in the above flow chart.

D. E. Krutz. Empathy-building interventions: A comparative analysis of experiential and expression-based interventions. In *Proceedings of the ACM/IEEE 46th International Conference on Software Engineering: Software Engineering Education and Training*, ICSE-SEET '24, New York, NY, USA, 2024. Association for Computing Machinery.

- [4] D. Mangano, S. Malachowsky, and D. Krutz. Supporting computing accessibility and artificial intelligence education using experiential learning labs. *Journal of Computing Sciences in Colleges*, 2024.
- [5] H. Moses, S. T. Thazin, S. Malachowsky, and D. Krutz. Experiential educational accessibility modules. In *Extended Abstracts of the 2023 CHI Conference on Human Factors in Computing Systems*, CHI EA '23, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9781450394222. doi: 10.1145/3544549.3574180. URL <https://doi.org/10.1145/3544549.3574180>.
- [6] W. Shi, S. Khan, Y. El-Glaly, S. Malachowsky, Q. Yu, and D. E. Krutz. Experiential learning in computing accessibility education. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Companion Proceedings*, ICSE '20, page 250–251, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450371223. doi: 10.1145/3377812.3390901. URL <https://doi.org/10.1145/3377812.3390901>.
- [7] W. Shi, S. Malachowsky, Y. El-Glaly, Q. Yu, and D. E. Krutz. Presenting and evaluating the impact of experiential learning in computing accessibility education. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Software Engineering Education and Training*, ICSE-SEET '20, page 49–60, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450371247. doi: 10.1145/3377814.3381710. URL <https://doi.org/10.1145/3377814.3381710>. Distinguished Paper Award.
- [8] W. Shi, H. Moses, Q. Yu, S. Malachowsky, and D. E. Krutz. All: Supporting experiential accessibility education and inclusive software development. *ACM Trans. Softw. Eng. Methodol.*, Sept 2023. ISSN 1049-331X. doi: 10.1145/3625292. URL <https://doi.org/10.1145/3625292>.

- [3] Y. Liu, D. Mangano, K. Neupane, S. Malachowsky, and

# Requirements Engineering Education with Just-in-Time Teaching and Projects

Ralf Reißing  
Coburg University of Applied Sciences  
Coburg, Germany  
ralf.reissing@hs-coburg.de

**Abstract**— Requirements Engineering (RE) is an important discipline of Software Engineering. Teaching RE should be based on two pillars: a solid methodological foundation and practical experience in application. This paper describes the teaching concept of a master's course on RE for engineers. The main components are Just-in-Time Teaching, exercises and mini-projects in small groups, and a big RE project with larger teams. So far, the course has been taught ten times. Experience is reported that led to the confirmation or to improvements of the teaching concept.

**Keywords**— requirements engineering education, just-in-time teaching, project work

## I. INTRODUCTION

The master's curriculum “Development and Management in Mechanical and Automotive Engineering” (MM) is located at the Department of Mechanical Engineering and Automotive Technology of Coburg University. MM has a flexible concept for the individual design of course content. Students can select their courses almost freely, allowing them to choose a broad or specialized course of study according to their own qualification goals [1].

The Requirements Engineering and Management (REM) course is an elective course with 5 ECTS credits (a total of 150 hours) for MM. The aim of the course is to qualify students with little prior knowledge to work independently in Requirements Engineering (RE) projects in the engineering domain.

Teaching practical RE well is a challenge. Inspired by Hagel et al. [2], the REM course was designed in 2014 based on Just-in-Time Teaching (JiTT) and practical tasks, and was continuously developed further [3][4]. The language used both for teaching and for documenting the requirements is German. The course has already been taught ten times.

## II. JUST-IN-TIME-TEACHING

JiTT [5] is a variant of Flipped Classroom. Teaching basic skills takes place not by lectures but in self-study using teaching materials provided (videos, texts, tasks, etc.). For preparation, students are guided by the learning objectives given in advance. After self-study, the classroom is used for answering open questions, for practice, and for consolidation. The strength of JiTT is that the content of the classroom time is adapted to the needs of the students at short notice based on feedback from the students (e.g. difficulties in understanding) and frequent mistakes in the submissions for the self-study tasks.

### A. Self-Study Phase

The first classroom activity of the week is preceded by a self-study phase consisting of two parts. The first part is a reading assignment on a chapter from the Sophists'

comprehensive RE book [6], which is available to the students for free as an e-book. For each reading task, there is a set of questions that help to focus attention on the major points when working through the chapter.

The second part of the self-study phase has four comprehension questions relating to the content of the reading task. The comprehension questions are mainly free-text questions in line with the higher competence levels of the learning objectives. Although free-text questions are much more time-consuming for the instructor to review, they are better highlighting students' errors in reasoning and difficulties in understanding.

There are three additional questions that are the same every week. The first one asks students about difficulties in understanding and open questions. The second question encourages students to reflect on the content (e.g. links to their own previous experiences, possible applications at work, further questions on content). The final question asks for the submission of a single-choice quiz question matching the content, with four possible answers. Students can earn bonus points by submitting sufficiently meaningful answers to the four comprehension questions and the quiz question.

### B. Classroom Phase

The instructor's task is to review the student answers submitted via the Moodle learning management system, give feedback in Moodle, and to design the first classroom session of the week accordingly. The classroom session starts with a selection of the submitted quiz questions as a revision. This is followed by impulses on the submitted difficulties in understanding and open questions. These stimuli are combined with general feedback on the answers submitted to the four comprehension questions. Examples of good solutions as well as examples of typical mistakes are selected from the submissions, presented, and discussed with the students. If there is still time, further examples or further content are provided.

## III. EXERCISES AND MINI-PROJECTS

The second classroom session of the week is used for the practical application of the skills taught. Students are given tasks to work on in small groups of 2-4 people and then present the results in the plenary discussion. The first tasks are to write down a small number of requirements for known applications such as cashless payment in the student's canteen, using Etherpad in Moodle. Later, practical applications of requirements elicitation techniques are added, e.g. field observation (observing real users) and apprenticeship (trying things out yourself) when operating the elevators in the university buildings. The groups then document their own requirements for elevator control and present them to the other groups.

An example of a mini-project is the review of a requirements specification using the formal inspection method. In the self-study phase, the students review the specification individually using a checklist provided. The review session then takes place in the classroom; moderated, recorded, and commented on by the instructor. Another mini-project uses peer assessment. Small specifications individually created by the students for the emergency call in an elevator are reviewed and commented on by two other students.

#### IV. REQUIREMENTS ENGINEERING PROJECT

In the second half of the semester, the second classroom session (so far for practical exercises) is used for the RE project. However, the teaching of content with JiTT in the first classroom session continues. In the RE project, students work on a large project task in teams of 4–6 people in a self-organized and practical way. The subject of the project is defined together with the students. All teams are given the same task but are allowed to adapt it individually in consultation with the instructor, for example by setting their own focus areas. The instructor supports the individual teams as a coach during the classroom session or in separately agreed consultation hours, both with regard to the RE approach and for assessing the quality of the intermediate results.

The most important learning objective of the RE project is to apply the skills acquired as independently as possible to a larger task. This involves choosing a suitable RE approach for the project as well as eliciting and documenting high-quality requirements. The project report and the requirements specification are submitted for grading. The report describes the organization of the team, the decisions on the RE approach (with justifications), the intermediate results, and the final result of the project. The students must also evaluate and reflect on all of this in their report (lessons learned).

Grading is based on the RE project report and on the team's requirements specification. Individual contributions for both the report and the requirements specification must be tagged so that individual grades for each student can be determined (as required by law). However, most teams opt all team members to be graded equally, as they equally contributed to the project's success. Bonus points acquired in the self-study phase can account for an extra of up to 10% of the basic total score. These bonus points improve the grade but are applied only if the exam is passed without them.

#### V. EXPERIENCES

On average, 10 to 15 students attend the REM course each year, allotted to 2–3 teams. There was an outlier in the summer semester 2020, when 28 students attended in 5 teams. This was due to the very spontaneous transition to online teaching ordered by the state government on the onset of the Covid-19 pandemic, just a few days before the start of the semester. While other MM courses had problems to get running for weeks, the teaching concept of the REM course could be transferred almost 1:1 to an online format. The two classroom sessions were shortened and converted into just one weekly 90-minute Zoom conference. During the RE project, a second coaching session via Zoom was offered.

In the course evaluations, students praise that they have learned a lot about RE – but also better German to be able to express themselves more correctly, consistently, and

precisely. The bonus points motivate them to work on the self-study tasks every week. The bonus points were introduced in the fourth year, as before there was frequent criticism that the effort put into self-study was not rewarded in the grade. However, there is still criticism of the large amount of time spent overall on the self-study phase.

The course is also very time-consuming for the instructor. For the JiTT part, 7 free-text answers per student must be reviewed and, if necessary, given feedback on for 12 weeks. An adequate schedule for the first classroom course must be derived from the student's feedback, and the necessary preparations, such as the procurement of teaching materials, must be made. Grading the submissions for the exam is also very time-consuming. One day must be set aside per team to review and comment on the report, the requirements specification, and any additional documents submitted, and to assess each team member individually. The students then receive qualified feedback via Moodle. It would be too exhausting for the instructor to offer all courses in this teaching format.

#### VI. CONCLUSION

Experience so far shows that self-study, JiTT, exercises, mini-projects, and the RE project complement each other well and thus result in a suitable teaching and learning concept. The RE project enables students to combine and deepen their skills acquired in the self-study phases, exercises, and mini-projects. They can test the suitability of their decisions on organization, methods, process models, and types of documentation for a practical problem. The students often learn the most from what goes wrong while trying.

An ongoing activity each year is to rework the self-study reading assignments and tasks to further reduce the work-load for the students while still achieving the learning outcomes. Every few years, a new edition of the Sophist's RE book is published, which requires further adjustments, but also keeps the course up-to-date.

#### REFERENCES

- [1] Coburg University, Development and Management in Mechanical Engineering and Automotive Construction, online: <https://www.coburg-university.de/academic-studies/degree-programs-in-german/technology-science-and-computer-science-master/development-and-management-in-mechanical-engineering-and-automotive-construction.html>, retrieved 11.02.2024.
- [2] G. Hagel, J. Mottok, and M. Müller-Amthor, “Three Feedback Cycles in Software Engineering Education by Extended Just-in-Time-Teaching“. In: A. Spillner, H. Licher (eds.): Software Engineering im Unterricht der Hochschulen (SEUH) 2013, pp. 17-26, Aachen, 2013. (in German)
- [3] R. Reißing, “Teaching and Examining in Requirements Engineering in an Engineering Master's Course.“ Workshop Studentische Ausbildung und Weiterbildung in Anforderungsspezifikationen (SAGWAS), Cottbus, October 2015. In: D. Cunningham, P. Hofstedt, K. Meer, I. Schmitt, I. (eds.): Informatik 2015. LNI 246, pp. 623-624, Bonn, 2015. (in German)
- [4] R. Reißing, “Using Just-In-Time-Teaching and Project Work in Requirements Engineering Education“, Softwaretechnik-Trends 42(3), pp. 18-20, 2022. (in German)
- [5] G. Novak, A. Gavrin, W. Christian, and E. Patterson, Just-In-Time Teaching: Blending Active Learning with Web Technology, Prentice Hall, 1999.
- [6] C. Rupp, and the Sophists: Requirements Engineering and Requirement Management, 7th edition, Hanser Verlag, 2021. (in German)

# Student Experiences in a Global Software Engineering Course Taught Jointly in Germany, Indonesia, and Japan

Felix Waigner\*, Andreas Rössler\* Michael Evan Santoso†,  
Patricia Brockmann\*, D. Moritz Marutschke†, Tri Astoto Kurniawan ‡ and Victor Kryssanov§

\*Computer Science Department, Nuremberg Institute of Technology, Nuremberg, Germany

†Faculty of Economics and Business, Kyoto University of Advanced Science, Kyoto, Japan

‡Faculty of Engineering, Brawijaya University, Malang, Indonesia

§College of Information Science and Engineering, Ritsumeikan University, Biwako-Kusatsu, Japan

**Abstract**—This poster paper explores the dynamics of a collaborative global software engineering (GSE) course, focusing on project management strategies in distributed settings involving teams from Germany, Indonesia, and Japan. An overview of the course organization and a real-world project to develop prototype solutions for smart tourism is given. This study highlights students' experiences working in multi-national cross-site teams. Difficulties encountered during the project phases, solutions devised and lessons learned by the students are presented.

## I. INTRODUCTION

Globalization has significantly contributed to the progress of project management methodologies in a vast majority of companies. Moreover, this globalization has not only provided a chance to exchange ideas between individuals, but it has also forced companies to adapt in order to meet the requirements of a world that is becoming increasingly complex. As companies seek talented individuals who can work on distributed projects, skills in global software engineering (GSE) are necessary [1]. Distributed projects, by their nature, are significantly more diversified, particularly in terms of time zones, geographical locations, and cultural backgrounds. This diversity presents unique challenges, such as communication barriers, coordination across different time zones, and cultural tolerance [2]. However, this diversity can also lead to new opportunities to examine problems from different perspectives, in order to generate improved solutions. This led to the development of courses about global software engineering at universities [3].

In this paper, we share our experiences participating in an international collaborative course in three countries: Nuremberg Institute of Technology (NIT) in Germany, Brawijaya University (BU) in Indonesia and Ritsumeikan University (RU) in Japan. This collaboration started in 2017, initially with two partners, NIT in Germany and RU in Japan [4], [5]. In the fall semester of 2023/24, the course was expanded to include BU in Indonesia as a third partner. The objective of the course is to expose students to global software engineering challenges, methods and technologies, focusing on project management.

## II. COURSE ORGANIZATION

The global software engineering class was a 15-week course spread over 3 months. The course began with introductory lectures conducted separately by professors at each university: NIT, BU and RU. International project management and challenges in intercultural communication were introduced.

The project assigned for this course was carried out in collaboration with the municipal government of Malang, which acted as the client. They served as the source of the requirements for the development of a smart tourism project.

Because the semester at BU, Indonesia started three weeks earlier than NIT and RU, the BU students were assigned to start early with the first phase of the project: elicitation of project requirements. On-site BU students gathered essential domain information to discover the challenges faced by city officials and future wishes for smart tourism in Malang.

Once the semesters at NIT and RU began, four cross-site teams were formed, each made up of students from BU, NIT and RU. Every week, a 90-minute video conference was held via Zoom, which included all of the teams and professors from all 3 universities. In addition, each team met separately to discuss their own project planning and implementation. Each team devised its own creative solution, which often included smart technology and machine learning. At the end of the semester, each team produced project documentation and presented a final prototype. One team was selected to present their solution prototype to Malang city officials.

## III. RESULTS AND LESSONS LEARNED

This section presents the results and the experiences of students from the BU, NIT and RU, who participated in the global software engineering course. First, communication was initiated within four cross-site teams, each consisting of students from various cultural backgrounds. Next, potential risks to the project timeline were identified, such as issues related to communication, culture, tools, roles, synchronization, and documentation. During the semester, the student teams devised solutions to address these risks, as discussed below.

#### A. Communication

Effective communication across geographic distance and diverse time zones was facilitated using tools like Zoom, Discord, and Google Drive. Initial engagement with informal small talk helped bridge cultural differences and fostered trust among team members. Long-term planning of communication has proved vital to react promptly to unexpected changes, ensuring ongoing communication throughout the project [6].

#### B. Cultural Awareness

Cultural awareness was fundamental for successful collaboration, guiding the team's communication, decision-making and problem-solving processes. Embracing diverse perspectives of cultures minimized misunderstandings and further enriched discussions. This created a collaborative environment that focused on shared goals rather than cultural differences.

#### C. Time Management

Effective management of time zone disparities (8 hours) was crucial for the teams' collaboration. Strategic scheduling and the use of project management tools facilitated efficient coordination across continents. This demonstrated the teams' commitment to inclusivity and transparency, fostering cohesion and shared responsibility, despite geographical distances.

#### D. Tools Used

For synchronous and asynchronous communication, the teams relied on Discord, using a plugin to address time zone differences. Miro's digital canvas aided in real-time collaboration on brainstorming, scheduling and design. Github served as the technical platform for code management, enabling efficient organization, version control and collaborative software development. Pair-programming sessions were conducted with GitHub Codespaces. Github also enabled continuous integration and continuous deployment (CI/CD). Visualizing complex data and processes was achieved with Draw.io. Google Workspace facilitated collaborative work on project documentation and presentations in real-time.

#### E. Clear Roles and Responsibilities

Establishing clear roles and responsibilities from the project's outset fostered purpose, accountability, and cohesion within the teams. Meticulous planning ensured alignment of tasks to team members' individual strengths, leading to streamlined workflow and efficient task execution. This clarity not only empowered team members to take ownership of their contributions, but also facilitated quick problem-solving by enabling individuals to know who to turn to for specific expertise or support when facing unforeseen challenges.

#### F. Flexibility and Adaptability

Resilience, flexibility and adaptability are essential in global teams[7]. The teams embraced an agile mindset to navigate unforeseen challenges, recognizing the diverse strengths and experiences within the team. Proactive adjustments, such as accommodating Christmas (Germany) and New Year's (Japan)

holidays, showcased the teams' understanding of different cultures and their collective efforts to enhance cohesion. Unforeseen problems, such as unstable internet connections, required flexibility of communication styles, collaboration tools and meeting schedules, to foster an environment where everyone felt empowered to contribute ideas for improvement.

#### G. Weekly Team Synchronization Meetings

Weekly team synchronization meetings, held outside of the official lectures, played a pivotal role in promoting consistent communication, information exchange and interpersonal connections among team members. These meetings provided a structured platform for sharing updates, addressing challenges and facilitating collaborative problem-solving. Video conferencing enhanced personal connections and provided non-verbal communication cues, fostering camaraderie and ensuring equitable access to necessary resources, thus promoting a unified approach to project tasks.

#### H. Documentation and Knowledge-Sharing

Processes, decisions and important deliverables were documented both as UML diagrams and in written form to ensure project continuity. Knowledge sharing was encouraged to leverage the diverse expertise within the international teams.

## IV. CONCLUSIONS

In summary, this work presents experiences and project management challenges encountered by students from NIT (Germany), BU (Indonesia), and RU (Japan), who took part in a hybrid, collaborative course on global software engineering. Students learned to employ cloud-based, collaborative communication tools to build bridges across time zones, geographical and cultural distances. They learned the value of weekly synchronization meetings, establishing clear roles and responsibilities. They devised flexible solutions to adapt to unexpected interruptions, such as different international holidays and unstable internet connections. Their experiences demonstrate the effectiveness of Project-Based-Learning.

## REFERENCES

- [1] I. Bosnić, I. Čavrak, and M. Žagar, "Assessing the impact of the distributed software development course on the careers of young software engineers," *ACM Trans. Comput. Educ.*, vol. 19, Jan 2019.
- [2] G. Hofstede, *Culture's consequences: International differences in work-related values*, vol. 5, sage, 1984.
- [3] P. Schmidtmayer, R. Chatley, J. P. Bernius, S. Krusche, K. Chaika, K. Krinkin, and B. Bruegge, "Global software engineering in a global classroom," in *Proceedings of the 44th International Conference on Software Engineering Education and Training*, pp. 113–121, 2022.
- [4] D. M. Marutschke, V. V. Kryssanov, and P. Brockmann, "East meets west: Global software engineering course in japan and germany," in *Proceedings of the 13th International Conference on Global Software Engineering*, pp. 30–34, 2018.
- [5] D. Marutschke, V. Kryssanov, and P. Brockmann, "Distributed virtual courses to teach global software engineering: Lessons learned and best practices," in *Proceedings of the 11th Intl. Conference on E-Education, E-Business, E-Management, and E-Learning*, pp. 256–260, 2020.
- [6] S. Morrison-Smith and J. Ruiz, "Challenges and barriers in virtual teams: a literature review," *SN Applied Sciences*, vol. 2, no. 6, pp. 1–33, 2020.
- [7] I. Čavrak, I. Bosnić, F. Ciccozzi, and R. Mirandola, "Resilience of distributed student teams to stress factors: A longitudinal case-study," *Information and Software Technology*, vol. 114, pp. 258–274, 2019.

# **Tool Demos**



# A Method and Software to Create SCORM Units for Computer Science Courses

Peter Braun

*Faculty of Computer Science and Business Information Systems*

*Technical University of Applied Sciences Würzburg-Schweinfurt*

Würzburg, Germany

peter.braun@thws.de

**Abstract**—Producing learning videos for flipped classrooms is a time-consuming and manual task. This paper presents a method and software to create SCORM units for computer science courses. The method consists of a common structure for learning videos, a Python script to create SCORM units, and ChatGPT for quiz creation. The software is based on the open-source software FFmpeg for video production. The results after 40 learning units show that producing a learning video can be automated to a great extent.

**Index Terms**—flipped classroom, learning units, video production

## I. INTRODUCTION

The flipped classroom is a teaching method that inverts the traditional teaching model [1]. In a flipped classroom, students watch video lectures at home to learn about a topic's foundations and deepen their knowledge when doing concept tests, exercises, or group discussions. The flipped classroom has been shown to improve student learning outcomes [2]. However, producing high-quality learning videos for flipped classrooms is a time-consuming and particularly manual task.

There are many ways to produce learning videos [3]: short whiteboard videos help students understand topics using comic-style animations. Lecture slides with narration but not showing the lecturer became popular during the Coronavirus pandemic and are still in use. Recording on-site lectures is also possible, but the videos often need to be more suitable for flipped classrooms, for example, because only the lecturer's voice but not the students' questions are audible in the recording.

We aimed to implement a flipped classroom in a 2nd-year Computer Science course, and we were inspired by the type of videos software companies like Google [4] used to explain their products to developers. In these videos, the speaker is in front of a green screen, and some slides are in the background. Often, live coding is used to demonstrate software development techniques.

We started producing this kind of learning video in 2019 without prior knowledge or expectation about the necessary effort. After the learning material (slides, exercises, quiz questions) was developed (pre-production phase), all videos for one learning unit were recorded in front of a green screen while reading text from a teleprompter (production phase). The post-production process was done with the video editing

software Camtasia [5] from TechSmith and included cutting the recordings, chroma-key the videos, placing the slides in the background, including interactive buttons, adding quiz questions, and finally producing a SCORM file to upload to a learning management system like Moodle. SCORM stands for Sharable Content Object Reference Model and is a collection of standards for e-learning content [6]. A SCORM file is technically a ZIP archive that contains a Web page with a connector to the hosting e-learning platform. For learning videos, the Web page includes a video player that can display subtitles for accessibility and additional content like quiz questions.

We also did experiments with other tools from Camtasia for animations and transitions. The post-production process took four to eight hours for a single 45-minute video. For all videos necessary for one semester, more than four weeks of full-time work were needed only for video production and post-production.

To reduce this amount of time, we first developed a common structure for all learning videos. This later helped create a Python script to automate the production of SCORM units. As a result, the post-production time was reduced to about 30 minutes per video with only a little manual work.

## II. UNIFORM STRUCTURE OF THE LEARNING VIDEO

We established a uniform structure for each video to efficiently produce a high volume of learning videos within a constrained timeframe. This approach streamlined the production process by limiting the choices in didactical techniques.

Each video, typically 45 minutes, comprises 20 to 30 distinct scenes. This uniformity allows various scene formats, such as the lecturer presenting in front of slides and screencasts featuring live coding sessions.

A video begins with an introductory scene displaying the university logo, followed by a title page with the lecture's title and the lecturer's name. Subsequently, the lecturer appears before a slide outlining the learning goals.

The learning unit's main content is split into topics of about 15 to 20 minutes. A topic commences with the lecturer imparting theoretical or background information using slides for knowledge transfer. The lecturer is shown in a half-length portrait in the right third of the screen. During screencast segments of live coding, the lecturer remains visible as a 'talking

head' in the lower right corner of the screen. Each topic within the video concludes with an exercise for students, at which point the video pauses, allowing students approximately 5 to 15 minutes to engage with the exercise before proceeding to the solution. A multiple-choice quiz also marks the end of a topic. A full learning unit spanning 45 to 60 minutes, typically includes two to three of these topics. Interactive elements are included in each topic, such as buttons for downloading additional materials or accessing git repositories and options to navigate between different scenes.

At the unit's conclusion, a recap of the learning goals is provided, followed by a slide with references and the university logo animation video.

### III. VIDEO PRODUCTION PROCESS

This section will describe the video production process in more detail. This must be seen in addition to general guidelines like [7]. Our process is supported by checklists developed and refined over the last few years. It comprises 13 phases with about 50 tasks to create one learning unit.

- 1) After the learning goals are defined, we continue by describing the contents of all exercises as they define the learning outcome of each topic.
- 2) Next, we create the slides and develop all source code examples shown on the slides and in the live coding scenes.
- 3) The most important and time-consuming phase is writing the teleprompter text, which is later used to record the video. We see this phase as the most important difference to only recording a slide show with narration. The advantages of writing a teleprompter text are that the content is complete and consistent, without any repetitions, the lecturer can look directly into the camera, and the text can be used as subtitles for accessibility.
- 4) The next phase is recording the video in the studio before the green screen. The content of every slide is recorded individually. There must be no or only very few tongue slips, as cutting out these from the video cannot be automated. In the beginning, a student assistant was operating the camera, and the teleprompter, but later, we connected the camera directly to the computer, used FFmpeg [8], [9] to record the video stream, and used a foot pedal to control the teleprompter.
- 5) Screencasts are recorded with Camtasia from TechSmith and spoken freely and without script. Later, in the post-production phase, the audio of the screencasts is transcribed using Whisper from OpenAI.

### IV. PYTHON SCRIPT TO CREATE SCORM UNITS

The output of the last phase is a set of video files, one for each slide, a PDF file containing the slides, and a text file containing the teleprompter text. The next phase is to create a SCORM unit from these files. We developed a Python script to automate this process. The script first splits the PDF file into single PNG files, one file for each slide. Then, it trims all video recordings so that at the beginning and the end of

each video, there is only one second of silence. This allows a smooth transition between slides. The script then merges the video recordings with the PNG files to create a video file for each slide. For all video editing tasks, the script uses the open-source software FFmpeg. Buttons for navigation between slides, accessing Web pages, or continuing the video after an exercise are added as images merged with the video.

The script then creates the SCORM unit, which is a ZIP file containing an HTML page with a video player that can show a table of contents to navigate between scenes and shows the teleprompter text as subtitles. In addition, the video player can also show quiz questions and evaluate the answers. The most difficult algorithmic task was synchronizing the teleprompter text with the video. The script uses a straightforward approach of assuming constant speaking speed and no longer breaks. All words spoken on one slide are shown in chunks of 20 words for an equal amount of time. The configuration files also contain information about buttons and quiz questions.

This script works completely automatically and can create a SCORM unit from the output of the production process. Depending on the computer's hardware, the script takes about 30 minutes for one learning unit. The SCORM unit is uploaded to the learning management system Moodle and briefly checked, e.g., if the table of contents is correct and the quiz questions are working. This is a manual task, but it takes only a few minutes.

### V. CONCLUSIONS

We have developed a method and software to create SCORM units for computer science courses. The method consists of a common structure for learning videos, a Python script to create SCORM units. The software is based on the open-source software FFmpeg. We used this method to create almost 40 learning units of 45 minutes each. The results show that producing a learning video can be automated to a great extent. Currently, we are working on automatically transcribing the screencast videos (because we do not use a script for recording them).

### REFERENCES

- [1] C. Reidsema, L. Kavanagh, R. Hadgraft, and N. Smith, Eds., *The Flipped Classroom: Practice and Practices in Higher Education*. Singapore: Springer Singapore, 2017, <http://link.springer.com/10.1007/978-981-10-3413-8>.
- [2] S. Delozier and M. Rhodes, "Flipped classrooms: a review of key ideas and recommendations for practice," *Educational Psychology Review*, vol. 29, 03 2017.
- [3] E. Navarrete, A. Nehring, S. Schanze, R. Ewerth, and A. Hoppe, "A Closer Look into Recent Video-based Learning Research: A Comprehensive Review of Video Characteristics, Tools, Technologies, and Learning Effectiveness," <http://arxiv.org/abs/2301.13617>, Aug. 2023.
- [4] "Flutter Youtube Channel," Google, <https://www.youtube.com/flutterdev>.
- [5] "Camtasia," TechSmith, <https://www.techsmith.com/video-editor.html>.
- [6] "SCORM 2004 overview for developers," Rustici Software, <https://scorm.com/scorm-explained/technical-scorm/scorm-2004-overview-for-developers/>.
- [7] S. Castillo, K. Calvitti, J. Shoup, M. A. Rice, H. Lubbock, and K. H. Oliver, "Production processes for creating educational videos," *CBE Life Sciences Education*, vol. 20, 2021.
- [8] "FFmpeg Tool," FFmpeg Developers, <http://ffmpeg.org/>.
- [9] S. Tomar, "Converting video formats with FFmpeg," *Linux Journal*, vol. 2006, no. 146, p. 10, 2006.

# Accessible Learning Labs: Accessibility Education Through Experiential Learning

Yang Liu\*, Xiaofan Que†, Dingrong Wang‡, Samuel A. Malachowsky§ and Daniel E. Krutz¶

Golisano College of Computing and Information Sciences

Rochester Institute of Technology

Rochester, NY, USA

Email: \* [yl4070@rit.edu](mailto:yl4070@rit.edu), † [xq5054@rit.edu](mailto:xq5054@rit.edu), ‡ [dw7445@rit.edu](mailto:dw7445@rit.edu), § [samvse@rit.edu](mailto:samvse@rit.edu), ¶ [dxkvse@rit.edu](mailto:dxkvse@rit.edu)

**Abstract**—Accessibility is a key aspect in ensuring the development of inclusive software. Unfortunately, research demonstrates that a large portion of software today is not created in an accessible manner. Problematically, students may not understand how to create accessible software, additionally misunderstanding the importance of creating accessible software. To fill the gap in accessibility education, we have created a comprehensive collection of labs, collectively referred to as *Accessible Learning Labs* (ALL). They have the primary objectives of educating participants on how to properly create accessible software, simultaneously illustrating the need to create inclusive and accessible software. In addition, the lab activities enable students to experience the implications of inaccessible software and make repairs based on their experience, making the software in the labs accessible. This will benefit many members of the software engineering community, ranging from beginning-level students to experienced practitioners who want to ensure that they are properly creating accessible and inclusive software. Complete project material is publicly available on the project website: <https://all.rit.edu>

**Index Terms**—Accessibility, Software Education

## I. BACKGROUND AND INTRODUCTION

Accessibility education is crucial for software engineers to create inclusive software. Without this knowledge, engineers can unfortunately develop software that excludes people with disabilities. Research shows that we continue to be deficient in creating inclusive and equitable software [1]. Although accessibility is a crucial computing topic, it is often excluded from education. To support the inclusion of this imperative computing topic, we need high-quality, easily adoptable material for our computing courses.

To address the issue of limited accessibility educational material in computing, we have developed a set of carefully designed educational modules known as the *Accessible Learning Labs* (ALL) [2]. These labs are designed to impart fundamental knowledge about accessibility concepts in computing and to foster awareness among participants of the importance of developing accessible software. The labs are easily adoptable as they are entirely web-based, requiring only a web browser without the need for software downloads or installations. Their self-contained nature and included comprehensive instructional materials allow for easy integration into diverse curricula, spanning from high school to undergraduate and graduate computer science courses. Our previous publications have demonstrated the effectiveness of experiential educational

interventions and the pedagogical effectiveness of ALL material and their format [3, 4, 5].

## II. LAB DESIGN

The Accessible Learning Labs (ALL) [2] are comprised of more than a dozen labs that focus on a variety of topics, ranging from how to create accessible software to foundational concepts in machine learning. Each lab consists of several components to help users understand a particular accessibility issue in software development. Reading material is provided to introduce the topic addressed, followed by an experiential exercise to enhance users' understanding of the topic. Additional materials and a quiz are offered to reinforce the learning objectives.

## III. DEMONSTRATED LABS

We will present several of our accessibility-focused labs.

### A. Accessibility to Sound and Speech

This lab explores the “Perceivable accessibility principle” by simulating scenarios that highlight the inaccessibility of audio cues for users who are Deaf/Hard of Hearing (D/HH) and prompting participants to propose changes to improve accessibility. Initially, understanding the significance, characteristics, and requirements of users within the Deaf and Hard of Hearing (D/HH) community, as well as accessibility standards, is established through validation and examination, such as the case study of the National Association of the Deaf (NAD) vs Netflix Lawsuit. The professional responsibility of a software engineer is to ensure the accessibility of web applications for people with hearing loss by incorporating alternative approaches to audio cues, in accordance with ethical standards and legal requirements. Then an experiential exercise is provided that demonstrates the importance of creating software that is accessible to Deaf/HH users (Figure 1) is provided.

### B. Accessibility for Dexterity Challenges

This lab provides an overview of dexterity accessibility challenges and their impact on software usage, presenting examples of web standards for dexterity accessibility and engaging users in exercises that simulate dexterity impairments to update code for improved accessibility. It first introduces dexterity accessibility challenges, impacts on software usage,

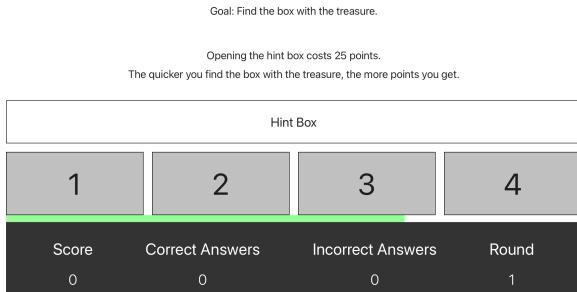


Fig. 1: The participants will experience playing the game both with and without sound effects in this exercise.

and accessibility standards. In the lab, the participant experiences enhanced accessibility designs for users with dexterity challenges.

#### C. Accessibility to Cognitive Impairments

This lab educates users about cognitive accessibility challenges, common cognitive impairments and inaccessible scenarios, and guides users to make changes to enhance accessibility for individuals with cognitive impairments. This lab first simulates cognitively-impaired reading scenarios and then designs various ways to help developers create more accessible software. Some manners include using appropriate headings and subheadings to reduce cognitive load, allowing users to have enough time to read, and providing clear descriptive feedback on forms, as shown in Figure 2.

#### D. Accessibility to Localization

This lab guides users through the localization process, enabling them to adapt software to accommodate different languages, cultures, and locales. In this exercise, students help a fictional university localize their website so that it is accessible to users in various locales.

#### IV. CONCLUSION

In this demo, we illustrate several selected labs we created to fill the gap in current software education needs in accessibility education. We also demonstrate how this can be used to educate participants on how to properly create accessible software, illustrating the need to create inclusive and accessible software. These interactive lab activities will help students in software engineering appreciate the importance of accessibility in software and gain concrete knowledge and practical skills to address challenges in inclusive software.

#### Acknowledgements

This material is based upon work supported by the NSF under grant #1825023, #2145010, and #2111152.

#### REFERENCES

- [1] R. K. Bellamy, K. Dey, M. Hind, S. C. Hoffman, S. Houde, K. Kannan, P. Lohia, S. Mehta, A. Mojsilovic, and S. Nagar. Think your artificial intelligence software is fair? think again. *IEEE Software*, 36(4):76–80, 2019.

#### COGNITIVE IMPAIRMENTS: EXERCISE

1.0 Dementia

Dementia is defined as a severe loss of cognitive abilities that disrupts daily life. Symptoms include difficulty remembering information, difficulty with organizing thoughts, difficulty working within time limits, and visual processing difficulties, which can affect the ability to recognize places. Content optimized for this group includes large, clear buttons with simple graphics and text, limited features, clear, step-by-step instructions, and rapid and direct feedback.

Seconds Left to Read: 0: 20 NEXT

(a) Simulated not accessible font style and size

1.0 Dementia

Dementia is defined as a severe loss of cognitive abilities that disrupts daily life

1.0.1 Symptoms

Common symptoms are:

- Difficulty remembering information
- Difficulty with organizing thoughts
- Difficulty working within time limits
- Visual processing difficulties, which can affect the ability to recognize places

1.0.2 Content optimized for this group

Content made for people with dementia tends to have:

- Large, clear buttons with simple graphics and text
- Limited features
- Clear, step-by-step instructions
- Rapid and direct feedback

Seconds Left to Read: 0: 06 NEXT

(b) Example of corrected font style and size

Fig. 2: The participant will experience how the improved accessibility of the software improves the user experience for dexterity impairments.

- [2] A. L. Labs. Accessible learning labs. <https://all.rit.edu>.
- [3] W. Shi, S. Khan, Y. El-Glaly, S. Malachowsky, Q. Yu, and D. E. Krutz. Experiential learning in computing accessibility education. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Companion Proceedings*, ICSE '20, page 250–251, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450371223. doi: 10.1145/3377812.3390901. URL <https://doi.org/10.1145/3377812.3390901>.
- [4] W. Shi, S. Malachowsky, Y. El-Glaly, Q. Yu, and D. E. Krutz. Presenting and evaluating the impact of experiential learning in computing accessibility education. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Software Engineering Education and Training*, ICSE-SEET '20, page 49–60, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450371247. doi: 10.1145/3377814.3381710. URL <https://doi.org/10.1145/3377814.3381710>. Distinguished Paper Award.
- [5] W. Shi, H. Moses, Q. Yu, S. Malachowsky, and D. E. Krutz. All: Supporting experiential accessibility education and inclusive software development. *ACM Trans. Softw. Eng. Methodol.*, Sept 2023. ISSN 1049-331X. doi: 10.1145/3625292. URL <https://doi.org/10.1145/3625292>.

# Automated Assessment Tool for Teaching Web Application Development

1<sup>st</sup> Basem Suleiman  
*Computer Science and Engineering*  
*University of New South Wales*  
Sydney, Australia  
basems@cse.unsw.edu.au

2<sup>nd</sup> Muhammad Johan Alibasa  
*Faculty of Engineering & Technology*  
*Sampoerna University*  
Jakarta, Indonesia  
johan.alibasa@sampoernauniversity.ac.id

3<sup>rd</sup> Anthony Wang  
*School of Computer Science*  
*University of Sydney*  
Sydney, Australia  
anthony.wan@sydney.edu.au

**Abstract**—This research paper proposes the design and implementation of automated testing tool specifically tailored for supporting the learning and training of web application development. The tool aims to streamline the assessment process for instructors by automating code marking for web application teaching and training tasks, reducing the time-intensive manual assessment process. Unlike existing tools that primarily target static pages or monolithic source code files, our tool addresses the dynamic nature of modern web development courses and training, incorporating modern web development frameworks. Utilising Vue.js, Node.js, and MongoDB, the tool offers a robust infrastructure for automated test case generation, execution, and feedback provision. This paper contributes to the training and educational technology field by enhancing automated grading and feedback efficiency, enabling accurate differentiation between student assessment work and standard answers, thus bridging gaps in existing tools reliant on line-by-line comparison algorithms.

**Index Terms**—automated marking, web application development, testing tool, education

## I. INTRODUCTION

The demand for efficient tools that streamline educational processes while enhancing learning outcomes continues to grow. With this need in mind, our project aims to develop an automated testing tool focused specifically on web application development learning and training tasks. Our primary goal is to build a solution that helps University educators and practitioners to automatically assess students' code on their web application learning tasks or projects. Manual marking on web app projects consumes considerable effort and time due to the setup process and manual testing of the features required by the various web application frameworks and configurations [1], [2]. Currently, most automated tools focus only on static pages or monolithic programming tasks [2]–[7]. However, web application development courses nowadays teach students and practitioners to use frameworks that require tedious setup and configurations, such as Node.js and other packages.

Our proposed tool contributes to the educational and training technology area by addressing the need for automated testing tools designed specifically for web application development learning. By streamlining the process of assessing students' code submissions, our tool alleviates the burden on instructors while fostering a more efficient and effective learning environment. In our tool, the primary task for data analysis revolves around testing and evaluating students' web

development learning tasks. Once both the answers (source code files) or student submissions are submitted, we proceed by assigning the answer and assignment to the JS library named Mergely. This tool assists us in analysing and evaluating the submissions, and subsequently, we process the results generated by Mergely to enhance their readability. Ultimately, we render these results to users through our front-end interface.

## II. RELATED WORK

A survey categorised the automated assessment into two primary categories: dynamic analysis and static analysis [8]. Dynamic analysis involves evaluating the attributes of a running program by determining which properties will hold for one or more executions [9], while static analysis tools assess software without executing it or considering inputs [10]. Numerous industry-standard static analysis tools have been incorporated into automated assessment for learning and training tasks, including linters such as pylint, CheckStyle, and others. However, most automated tools for web application training are performed statically, such as through parsing the submitted code. Developing automated tools for assessing web application tasks presents challenges, as it requires setup and resources to execute the code at scale.

Recent systematic literature review [2] has showed extensive research on automatic grading tools for programming courses. However, these reviews indicated that the majority of studies focused on Object-Oriented Programming (OOP) grading, with significantly fewer studies focusing on web-related assessments. One of the studies, Nguyen et al. [4] introduced a tool to assess the quality of web-based team projects. The authors employed continuous integration tools to analyse version control logs to ascertain students' contributions and utilised existing static analysis tools, including SonarQube and StyleLint, to evaluate the source code's quality. However, this tool primarily emphasises static analysis rather than executing the submissions of web applications.

## III. METHOD

In our tool, the main data analysis step involves testing web development learning tasks. Once the standard and answers are submitted, we employ a JavaScript library called Mergely to compare them. Then, we process the results to make them

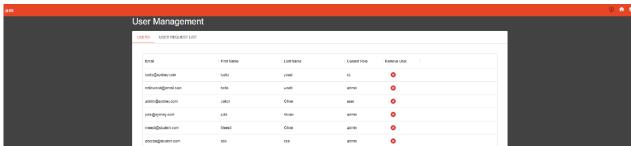


Fig. 1. User management page

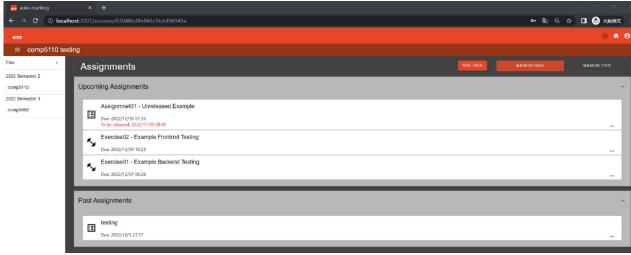


Fig. 2. Task or assignment page

easier to understand before showing them to users on our front-end interface.

In this project, we employed various testing methods to ensure high-quality software development, including Unit Testing, Integration Testing, and End-to-End Testing. End-to-End testing involved recruiting five users with varying permissions (participants or student, trainer, admin) to simulate real-world interactions. Participants were tasked with completing predefined user goals and tasks independently, while their reactions and feedback were recorded. Any encountered issues were addressed, and failed test cases were fixed and retested to ensure overall system functionality.

#### IV. RESULT

We conducted a thorough analysis of the requirements provided by the stakeholders and identified the key functions and features necessary for the automated testing tool. Based on this analysis, we created a design specification outlining the system architecture, user interface, and interactions. Employing an iterative and incremental approach, we continuously refined and improved the prototype based on feedback and evaluation. The prototype was developed utilizing a technology stack that included JavaScript, Vue, Element UI, Express, Node, Mongoose, and MongoDB.

The automated testing tool was designed to simplify the testing process for web applications implemented in JavaScript, HTML, and CSS. It facilitates automated test case generation

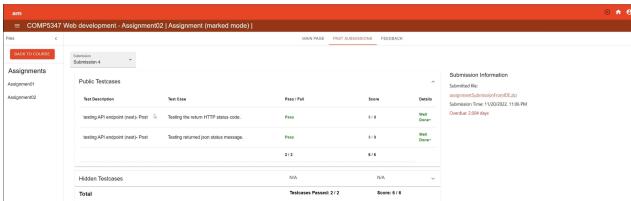


Fig. 3. Automated test result page

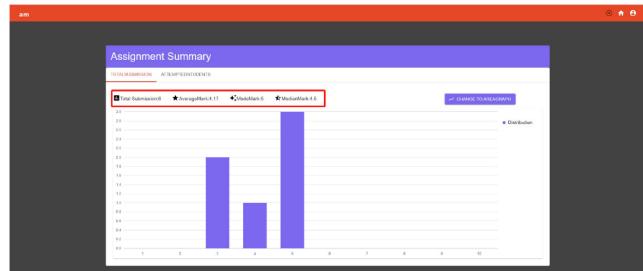


Fig. 4. Result summary page

and execution, along with providing automated feedback on web application exercises and tasks. The tool encompasses several main features, including user management (Fig. 1), course management, student assignment submission (Fig. 2), and automatic test case creation and execution (Fig. 3 and 4).

#### V. CONCLUSION

Our findings showed the importance of automating and simplifying the web application testing process, thus saving time and effort for instructors. The tool addresses a gap in the field by offering an integrated solution for test case generation, code evaluation, and performance analysis. Our findings and recommendations carry significant implications for the field of automated detection. Currently, most automated testing software or code training platforms on the market rely on line-by-line comparison algorithms to test participant-submitted learning tasks. However, our project facilitates efficient automated assessment by generating test cases capable of accurately discerning differences between participant-submitted assignments and the standard answers provided by instructors.

#### REFERENCES

- [1] H. Aldriye, A. Alkhafaf, and M. Alkhafaf, "Automated grading systems for programming assignments: A literature review," *International Journal of Advanced Computer Science and Applications*, vol. 10, no. 3, 2019.
- [2] M. Messer, N. C. Brown, M. Kölling, and M. Shi, "Automated grading and feedback tools for programming education: A systematic review," *ACM Transactions on Computing Education*, vol. 24, no. 1, pp. 1–43, 2024.
- [3] A. Järventausta, "Piloting a new automated assessment tool in the web applications course," *LUT University*, 2022.
- [4] B.-A. Nguyen, K.-Y. Ho, and H.-M. Chen, "Measure students' contribution in web programming projects by exploring source code repository," in *2020 Intl. Computer Symposium (ICS)*. IEEE, 2020, pp. 473–478.
- [5] X. Fu, B. Peltzverger, K. Qian, L. Tao, and J. Liu, "Apogee: automated project grading and instant feedback system for web based computing," *ACM SIGCSE Bulletin*, vol. 40, no. 1, pp. 77–81, 2008.
- [6] M. Sztipanovits, K. Qian, and X. Fu, "The automated web application testing (awat) system," in *Proceedings of the 46th Annual Southeast Regional Conference on XX*, 2008, pp. 88–93.
- [7] L. Zamprogno, R. Holmes, and E. Baniassad, "Nudging student learning strategies using formative feedback in automatically graded assessments," in *Proc. of the 2020 ACM SIGPLAN Symp. on Splash-E*, 2020.
- [8] K. M. Ala-Mutka, "A survey of automated assessment approaches for programming assignments," *Computer science education*, vol. 15, no. 2, pp. 83–102, 2005.
- [9] T. Ball, "The concept of dynamic analysis," *ACM SIGSOFT Software Engineering Notes*, vol. 24, no. 6, pp. 216–234, 1999.
- [10] N. Ayewah, W. Pugh, D. Hovemeyer, J. D. Morgenthaler, and J. Penix, "Using static analysis to find bugs," *IEEE software*, vol. 25, no. 5, 2008.

# A Tool to Facilitate Calibrated Peer Reviews in Software Engineering Education

Bastian Tenbergen

*Department of Computer Science  
State University of New York at Oswego  
Oswego, New York, United States  
[bastian.tenbergen@oswego.edu](mailto:bastian.tenbergen@oswego.edu)*

**Abstract**—In previous work, we have provided a substantive amount of evidence that shows that Calibrated Peer Reviews to facilitate a feeling of “quality” in software engineering students have a strongly positive impact on student learning. Peer reviews repeatedly expose students to examples of different quality and force students to articulate engineering tradeoffs. Using calibration examples of different quality guides students in their evaluation of peer solutions and fosters an appreciation between good and bad solutions to exercise problems in requirements engineering and testing. The result is a significant improvement in test scores and theory retention, however, the overhead for the instructor to facilitate the peer review process outweighs the time saved for grading. In this article, we introduce “CPRtool” to automate collection and distribution of student solutions and peer feedback.

**Keywords**—*Quality Assurance, Requirements Engineering, Software Engineering Education, Calibrated Peer Review, Peer Assessment, Peer Learning*

## I. INTRODUCTION

In recent years [1], instructional approaches involving peer reviews and peer assessment have become a popular and effective means to increase student motivation, retention, and soft skills [2], especially in software engineering (SE) education. The key difference to “traditional” skill-based instruction is that students not only complete project or homework assignments, which are then graded by an instructor, but students are also involved in the assessment of other students’ solutions. In particular, Calibrated Peer Reviews (CPR) have shown promising results in theory-heavy Software Engineering (SE) instruction [3]. CPR [4] is a formative peer-assessment technique [5], where learners make use of instructor-provided example solutions of different maturity levels to evaluate solutions created by their peers. Paired with a grading rubric, the idea is to assist students in identifying engineering tradeoffs and distinguishing solutions of different degrees of quality. CPR have shown an effective means to improve nearly all aspects of instruction in many different fields [6]-[8].

In previous work, we have shown that CPR have a significantly positive effect on summative learning activities (i.e., assignment sheet scores<sup>1</sup>) as well as theory retention (i.e., exam grades) in a requirements engineering course [9] and, to a lesser degree, in a quality assurance course [10]. In both courses,

results show that CPR increase theory retention and improve students’ “feeling of quality”, as assessed by their ability to articulate engineering decisions and discriminate poor from adequate engineering trade-offs. We have furthermore shown how CPR can be used in an integrated pedagogical approach that repeatedly exposes students to solution alternatives in the absence of ideal, perfect solutions. Therein, students are systematically acclimated to identifying and articulating engineering tradeoffs. However, the drawback of our approach is that while CPR alleviates some of the burden to grade student solution to assignment sheets on part of the instructor, some of the time-benefit especially in large courses is outweighed by the overhead of facilitating the CPR process.

To alleviate administrative overhead, this paper introduces tool support for CPR. CPRtool is a microservice web application to distribute assignment sheets, collect student responses, and facilitate the peer review process with scoring rubrics. In the following section II, we describe the manual process this tool is meant to facilitate. In the final section III, we describe the rough architecture of the tool, which is available free of charge at: <https://github.com/tenbergen/CPRTool>.

## II. CALIBRATED PEER REVIEWS IN SE INSTRUCTION

Calibrated Peer Reviews have been applied in the RE course starting in 2021 [9] and in the QA course starting in 2022 [10]. Both courses follow the Think-Pair-Share principle [11], in which students are first exposed to theory and classroom examples before they are asked to solve problems on assignment sheets on their own. The assignment sheet preparation period is typically 10 days long, as shown in Fig. 1. After this period, solution strategies are discussed in class and anonymized solutions are collected from each team of typically two students. Solutions do not contain names, but unique-to-them team designations to minimize bias. Each team then reviews solutions of other teams. An effort is made to assign reviews such that each team solution receives roughly the same number of reviews (with a target of at least three), and each team is asked to review roughly the same number of other teams’ solutions (also with a target of at least three). Unequal effort is balanced such that each solution receives the same amount of feedback, and every student has the same review burden across the semester. When reviews are assigned, a team-specific “review package” is distributed containing other teams’ solutions to be reviewed

<sup>1</sup>We use the term “grade” to refer to an evaluation done by an instructor, while we use the term “score” to refer to points awarded or deducted by student peers.

along with high-quality, medium-quality, and low-quality examples from previous semesters, the instructor's "ideal" solution, and a scoring rubric with feedback template. This review package meant to "calibrate" the peer evaluation process by providing an evaluative framework. After a review period of approximately one week, feedback is collected, collated for each deanonymized team (i.e., the instructor maintains a mapping of student names to team designations such that feedback can be returned to the right teams). Before returning the feedback to the students, the instructor then checks the quality of the reviews against evidence of too harsh or too lenient scoring as well as, e.g., vagueness or profanity in the review comments.

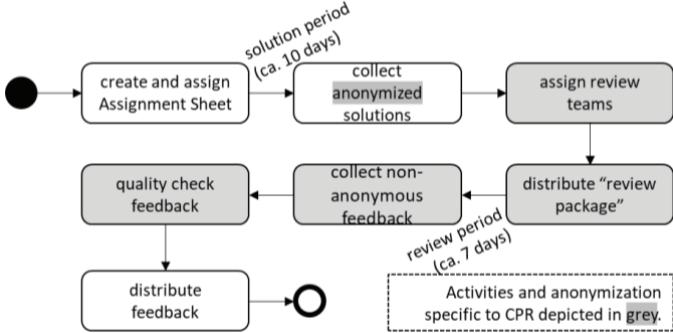


Fig. 1 Calibrated Peer Review process in both courses.

### III. TOOL SUPPORT

CPRtool was developed by students in the "Software Design" course instructed by the author, as described in detail in a manuscript in this very volume [12]. The tool is work in progress and in a pre-release stage. Collaboration is welcome. The rough architecture is shown in Fig. 2 below.

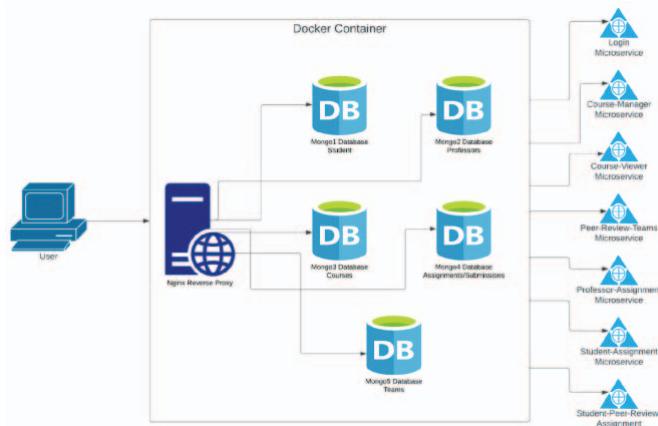


Fig. 2 Calibrated Peer Review process in both courses.

CPRtool consists of seven independent microservices deployed in dedicated docker containers. An eighth docker container running an Nginx reverse proxy orchestrates access between the microservices and their MongoDB databases. The **Login Microservice** authenticates users. Instructors are differentiated from students based on the signature of their email address, but any user can be elevated to instructor privileges. The **Course Manager Microservice** allows instructors to add, remove, and edit courses as well as to edit student rosters. Edit

team names and team compositions is done using the **Peer Review Teams Microservice**, which has a student-facing and instructor-facing front-end to allow editing one or all teams, depending on the user's role. The **Course Viewer Microservice** allows students to view courses they are enrolled in and access assignments instructors have created therein. Instructors create assignments using the **Professor Assignment Microservice**, which allows adding assignment instructions and templates, editing due dates, etc. For each assignment that is subject to peer review (i.e., there is no requirement to pair every assignment with peer reviews – it is possible to assign homework for traditional grading by the instructor as well), instructors can add the review packages including the calibrated solutions discussed above to each assignment. The **Student Assignment** and **Student Peer Review Assignment** microservices provide front-end components to allow students to view instructions, due dates, etc. for assignments and their respective peer reviews, and upload solutions accordingly. The **Student Peer Review Assignment Microservice** furthermore conducts profanity checks, calculates grades based on all submitted peer review feedback for any given team, and alerts instructors of outliers or grades using a separate **Email Microservice** (not in Fig. 2).

### ACKNOWLEDGMENT

We thank our students of the Fall 2022 and Spring 2023 cohort of our Software Design course and for implementing CPRtool.

### REFERENCES

- [1] M. Daun, A. M. Grubb, B. Tenbergen, "A systematic literature review of requirements engineering education." Requirements Eng 2022. <https://doi.org/10.1007/s00766-022-00381-9>
- [2] L. Likkel, "Calibrated Peer Review Essays Increase Students Confidence in Assessing Their Own Writing." J College Science Teaching 41(3), 2012, pp. 42-47.
- [3] M. Aniche, F. Mulder, F. Hermanns, "Grading 600+ Students: A Case Study on Peer and Self Grading." Proc 43<sup>rd</sup> Int. Conf Soft Eng: Soft Eng Edu & Train, 2021.
- [4] R. Robinson, "Calibrated Peer Review™ An Application to Increase Student Reading & Writing Skills. The American Biology Teacher 63(7), 2001.
- [5] W. Harlen, M. James, "Assessment and Learning: Differences and Relationships between Formative and Summative Assessment." Assessment in Edu: Principles, Policy & Practice 4(3), 1997, pp. 356-379.
- [6] S. Balfour, "Assessing Writing in MOOCs: Automated Eassay Scoring and Calibrated Peer Review™." Research and Practice Assessment 8, 2013, pp. 40-48.
- [7] M. Walvoor, M. Haefnagels, D. Gaffin, M. Chumchal, D. Long, "An Analysis of Calibrated Peer Review (CPR) in a Science Lecture Classroom." J College Sci Teach 37(4), 2008, pp. 66-73.
- [8] B. Furman, W. Robinson, "Improving Engineering Report Writing with Calibrated Peer Review™." Proc. 33rd Symp Ameri Soc Eng Edu, 2003.
- [9] B. Tenbergen, M. Daun, "Calibrated Peer Reviews in Requirements Engineering Instruction: Application and Experiences." Proc. 55<sup>th</sup> Hawai'ian Int'l. Conf. on Sys Sci, 2022. <http://hdl.handle.net/10125/79437>
- [10] B. Tenbergen, M. Daun, "Calibrated Peer Reviews in Software Engineering Instruction: Experiences and Applications." Proc. 35<sup>th</sup> IEEE Int'l. Conf. Software Engineering Education & Training, 2023.
- [11] M. Kaddoura, "Think pair share: a teaching learning strategy to enhance students' critical thinking." Edu Research Quarterly, 36(4), 2013.
- [12] B. Tenbergem, N. Mead, "Technology Acquisition Plans to Foster Supply Chain Risk Management Learning Outcomes in Project-Based Software Development Courses." Proc. 36th Int'l. Conf Software Enginnering Education & Training, 2024.

# **Industry Track**



# Can LLMs Facilitate Onboarding Software Developers? An Ongoing Industrial Case Study

Maider Azanza, Juanan Pereira and Arantza Irastorza

University of the Basque Country UPV/EHU

Donostia-San Sebastian, Spain

Email: {maider.azanza, juanan.pereira, arantza.irastorza}@ehu.eus

Aritz Galdos

LKS Next

Mondragon, Spain

Email: agaldos@lksnext.com

**Abstract**—Onboarding new software developers presents persistent challenges for teams, with newcomers facing steep learning curves and senior staff burdened by providing training and mentoring. This research explores leveraging *Large Language Models (LLMs)* to streamline onboarding, mitigating productivity losses from disrupted workflows. We collaborated with LKS Next, an IT consulting firm that has experienced first hand the challenges onboarding new developers brings. This paper presents an ongoing case study within LKS Next, conducted using action design research methodology. Two cycles have been completed, gathering feedback from newcomers, team leaders, and managers on issues like the need to prioritize self-directed learning resources over "burdening" mentors, and considerations around using third-party LLMs. The third cycle will focus on evaluating open source LLMs to maintain control within the company. Our envisioned goal is to develop an LLM-powered conversational agent, delivering tailored onboarding support while avoiding privacy risks.

## I. INTRODUCTION

Onboarding new employees is a critical but often challenging process for software teams. Newcomers face steep learning curves as they assimilate extensive information on coding standards, processes, and project details [1]. This overload can hinder grasping the big picture and effectively contributing to the team [2]. Moreover, onboarding requires substantial time from senior developers, who take time away from regular tasks to train and mentor. This temporary shift can negatively impact overall productivity of the entire team during onboarding periods [3].

Supportive practices like mentorship and prompt question answering help newcomers overcome onboarding obstacles but further tax senior developers [4]. This has been the experience of LKS Next, which is a cooperative consultancy firm with over 30 years of history. With shortage of talent to handle new projects, the firm needs to train junior developers on both project requirements and tools. Thus, onboarding of junior developers has become a major concern.

*Large language models (LLMs)* hold promise for enhancing onboarding through personalized and instant guidance tailored to newcomers' needs [5]. Thus, LKS Next decided to test their promise in their own setting. In order to do so, we carried out a case study using *Action Design Research (ADR)* [6] in an iterative way.

In the first cycle, we developed *I Need a Hero (INAH)* as an initial step. INAH was a prototype that matched newcomer

questions to expert contacts using OpenAI's GPT-3.5 language model [7]. However, an evaluation with newcomers clearly showed that asking more senior developers was deemed as the last resort, newcomers vastly preferred first trying to locate resources that could help.

Hence, the second iteration focused on enhancing INAH with access to other resources such as company documentation, courses, and previously answered questions. When INAH v2 was presented to managers, its reliance on this proprietary external LLM raised serious data privacy concerns. Commercial LLMs like ChatGPT<sup>1</sup> or Claude<sup>2</sup> are owned and operated by outside entities. Thus, using them would require sharing potentially sensitive internal company information, such as employee expertise, project details, and institutional knowledge, with a third party.

With that concern in mind and the aim at avoiding privacy risks associated with external dependence, a common occurrence among firms, the third cycle of this work focuses on the use of open source LLMs, specifically feeded on premise with the company's own data (i.e., LKS Next's data, including past projects and courses).

This paper details an ongoing industrial case study that explores the capability of LLMs to streamline the onboarding process for software developers at LKS Next. In Section II we provide insights into the company's onboarding context and the challenges newcomers face. The methodology and promise of employing LLMs to address these challenges are outlined in Section III, where we introduce our ADR approach. The iterative ADR process is dissected into three cycles, detailed in Sections IV, V, and VI. We conclude with a discussion on related work in Section VII and summarize our findings and their implications for future research in Section VIII. This research sheds light on the potential benefits and hurdles of integrating LLMs into software developer onboarding, offering lessons learned from the experience in LKS Next.

## II. ONBOARDING IN LKS NEXT

As introduced earlier, LKS Next is an independent consulting firm with over 30 years of experience onboarding new developers. After reflection, they identified four key

<sup>1</sup><https://chatgpt.com/>

<sup>2</sup><https://claude.ai/>

groups impacted by and presenting opportunities to improve onboarding:

- *Newcomers* must quickly acquire company procedures, technologies, tools, and domain knowledge. They also need clarity on their role, responsibilities, and expectations while building relationships and immersing in the company culture.
- Existing *team* members must share knowledge and best practices for the newcomer's tasks, which can add workload. Introducing a new member can also potentially strain team cohesion, especially with hybrid work, where teams combine face-to-face and remote days.
- *The IT business branch* managers play a key role in ensuring consistent onboarding, and leading to successful integration and long-term retention and performance. As a case in point, there are aspects that transcend individual teams and pertain to all IT operations. Examples include code management tools like GitLab<sup>3</sup>, concepts related to DevOps, or the utilization of tools such as Sonarqube<sup>4</sup>. Streamlining the onboarding procedures related to these common elements can result in increased efficiency and reduced workloads for each team.
- Aligning onboarding with *LKS Next's cooperative culture, values, and objectives* is crucial. Newcomers should understand the social initiatives around equality, environment, and the promotion of the Basque language.

To assess perceptions of the onboarding process, a survey was conducted within the firm. Interestingly, newcomers ( $n=72$ , onboarded in the past year) rated the process very positively, averaging over 4/5 on all survey items. However, team leaders ( $n=26$ , with onboarding experience) disagreed and rated the onboarding process as complex and taxing. Their main complaints centered on the substantial effort required for successful training and mentoring of newcomers, which prevented them from fulfilling other duties. While onboarding was overall deemed successful, it appeared to excessively tax senior developers and leaders. This last finding presents an opportunity for the firm to refine the onboarding process, easing the burden on senior profiles while maintaining the content, critical for newcomer integration and performance.

This is consistent with the existing literature. Not surprisingly, a factor that has been shown to be determinant in overcoming onboarding barriers is having an adequate support [8], [9]. Newcomers need to perceive that they can receive guidance and ask questions without evoking a sense of embarrassment [9].

In the pursuit of reaching such a helpful atmosphere, a strategy that has proven successful is appointing mentors [10], [11], [12], [13], [14], [15]. Having a reference person to ask questions, "dumb" as they may sound, reduces the barriers that hinder help seeking [16]. Mentors improve the productivity of newcomers [13] and allow them to adapt more easily to the practices of the organization [17]. Furthermore, mentors

function as central conduits through which newcomers can acquaint themselves with other members of the organization and gain insights into their respective roles. This allows newcomers to identify to whom to direct their inquiries in the future [3].

Nevertheless, this support and this mentoring strategy comes at a cost [18], [19]. While they are helping newcomers, senior developers are not performing their own work, thus further delaying the project for which they needed help in the first place [3].

In this context, LLMs can support organizations in guiding newcomers and increasing their independence during the onboarding process.

### III. LEVERAGING LLMs FOR ENHANCED ONBOARDING: AN ADR CASE STUDY

Artificial Intelligence has huge potential to aid in the onboarding processes [5]. Specifically, we advocate for LLMs to become tailored mentors to support newcomers in evolving into full team members.

LLMs are sophisticated machine learning systems trained on massive datasets to comprehend and generate natural language. These models are capable of contextual understanding, allowing them to engage in coherent, contextually relevant conversations and produce precise, coherent text [20].

In the short time they have been available, LLMs have revolutionized teaching [21], [22], [23]. While using LLMs presents risks and challenges [21], they have shown promise in different areas [23]. Regarding software development, their use is also being researched [24].

Given their capability as tutors that can be personalized, LKS Next believes that LLMs can provide real-time support and answers during onboarding, mitigating fear of interrupting colleagues and fostering an environment where queries are promptly resolved. However, similar to the teaching setting [22], [23], LLMs should complement, not replace, human interactions with other members of the team and the organization, which remain essential for successful onboarding [10], [11], [12]. The role of the LLMs is to improve and simplify the training process, making sure new developers move into their jobs smoothly, thereby reducing the invested effort of the organization.

Hence, we set out to analyze the potential of LLMs to reduce the onboarding effort in LKS Next. Specifically, if we follow the template proposed by Wieringa for design problems [25] we intend to:

- *Improve* the access to relevant information of newcomers during their onboarding process in software development teams
- *by* feeding an LLM with company information
- *that satisfies* answering questions and also providing ordered resources and contact information of knowledgeable colleagues
- *in order to* make newcomers more independent and increase their satisfaction, while also reducing the invested effort and generated frustration of senior developers.

<sup>3</sup><https://about.gitlab.com/>

<sup>4</sup><https://www.sonarsource.com/products/sonarqube/>

We draw from real-world practice in an organizational context, with focus on the artifact (the LLM). Our close collaboration with the firm allows us to employ a methodology that leverages the expertise of both researchers and practitioners. Hence, we chose *Action Design Research (ADR)* [6].

ADR combines two complementary approaches, namely *Action Research (AR)* and *Design Science Research (DSR)*. AR contributes co-creation involving researchers and practitioners, and iterative intervention cycles [26]. Conversely, DSR emphasizes artifact design as the primary driver and outcome of the research process. Hence, ADR is a method that focuses on the building, intervention, and evaluation of an artifact that addresses the class of problems typified by a specific situation, and that reflects the influence of users and ongoing use in context [6]. It is an iterative process.

The following section presents the first ADR cycle, directed at helping newcomers find the more senior colleague who can aid them with their questions.

#### IV. FIRST CYCLE: "I NEED A HERO"

In the firm, knowledge management issues can arise even in experienced teams, where identifying the right colleague to consult can be difficult. Note that LKS Next employs over 500 developers across multiple locations. This exacerbated in the case of newcomers, as they lack institutional knowledge and familiarity with company culture [8], [3].

To address this problem, we developed an initial prototype called *I Need a Hero (INAH)* to showcase the potential of LLMs for expertise location [7]. By leveraging on LLMs, INAH aims to connect employees to the right collaborators and mentors. When a newcomer has a question, INAH provides contact details for the in-house expert best equipped to address their specific inquiry [27]. The INAH prototype is implemented in Python and utilizes OpenAI's *ChatCompletion API*<sup>5</sup> to communicate with the GPT-3.5 language model.

Broadly speaking, INAH works in two different phases.

*Hero creation.* The initial phase involves the creation of the "heroes" within the system. This entails uploading the curriculum vitae (CV) of company employees, who are ready to help newcomers. Their text is extracted and three requests are made to GPT-3.5 to elicit: (1) contact data (currently name, telephone and email), (2) an array of the skills the hero has, based on their CV and, (3) an avatar of the hero. All these data are stored in a database.

*Looking for a hero.* When a newcomer submits a question to INAH, a request is made to GPT-3.5 to extract the required skills to address it. To identify heroes possessing these skills, INAH compares the extracted skills against those stored in the database. It then presents the newcomer a curated list of potential heroes and their contact information, as shown in Figure 1. This allows the newcomer to directly contact them.

This approach not only minimizes the need for newcomers to approach other colleagues as intermediaries, but also ensures that queries are directed to the most knowledgeable

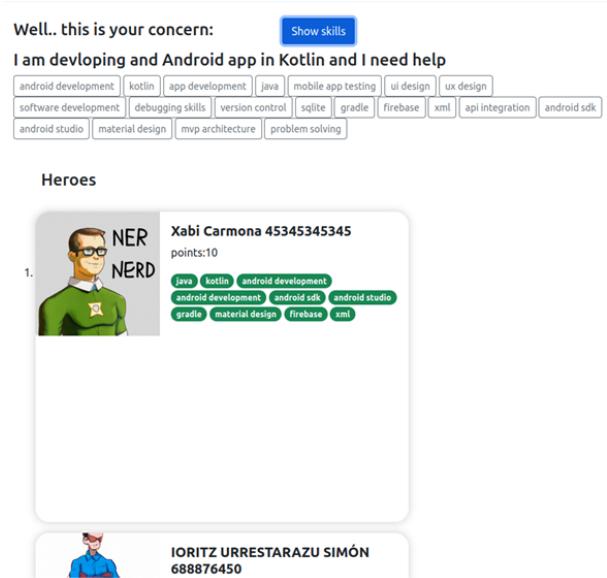


Fig. 1. INAH provides the list of heroes that can help the newcomer.

person, even if the newcomer is unfamiliar with that individual. Consequently, it reduces interruptions to more experienced teammates, fostering a work environment where newcomers can comfortably seek guidance with the assurance that their questions will be directed to the appropriate person.

During this first cycle, INAH underwent testing using fabricated data. It was then introduced to various members of the "technical-office", a department that provides cross-functional knowledge to other more specialized departments, to assess its efficacy in enhancing the onboarding process. This evaluation encompassed a diverse set of participants, including team leaders and a focus group of four newcomers who had been with the company for durations ranging from two to ten months. A focus group is an appropriate tool for gathering opinions and perceptions from a group of people about a particular topic, product or service. Participants can discuss and share their points of view on the tool, which can provide useful insights into its utility [28].

This evaluation provided valuable insights and feedback and showed potential for improvement. Two main topics emerged time and again. First, the consensus among newcomers was that the act of posing questions should be regarded as a final option. They expressed a preference for searching documentation, reviewing prior projects, or exploring online resources before resorting to direct inquiries.

**Lesson 1:** The results of the LLM should prioritize other resources that allow the newcomer to investigate independently. Contacting seniors is felt as a last resort.

The second topic concerned the ranking algorithm for heroes (subject matter experts) in the application. The current method prioritized heroes based on the number of shared skills with the question. However, this approach has limitations.

<sup>5</sup><https://platform.openai.com/docs/guides/chat>

The screenshot shows a user interface for a question-and-answer application. At the top, there's a search bar with the query "What are the holidays in LKS Next for the Basque autonomous community?". Below it, a note says "Aritz Galdos Otermin". A large green button labeled "A" is on the left. Below the search bar are buttons for "Heroes", "Similar questions", and "Direct help". A red banner at the top says "Watch out! This is an auto-generated answer. Use with caution." The main content area has a section titled "Description" with the text: "according to the page content, the holidays in lks next for the basque autonomous community are:" followed by a list of 13 holidays. Below the list, it says "that's a total of 13 holidays for the basque country in 2023 according to the provided information." There's a "Source" link to "uploads/ks/1-\_Normas\_Laborales\_2023\_LKS\_SCOP-1.pdf" and a "Download" button.

Fig. 2. INAH now provides the answer and the link to the pertinent document. Related questions and heroes can also be accessed.

Certain skills may be more relevant than others, skills can become outdated, and those with extensive expertise often have less availability to mentor newcomers.

Therefore, the criteria for hero prioritization in the system warrants reevaluation and refinement. Beyond technical alignment, additional factors like seniority, availability, and interpersonal rapport should influence matchmaking. While top experts possess valuable knowledge, newcomers may feel more comfortable approaching peers or friendly colleagues for guidance. In summary:

**Lesson 2:** When recommending colleagues to newcomers, considerations beyond technical expertise are important. Newcomers may prefer turning to approachable peers or colleagues they have existing rapport with, rather than busier top experts.

This insights led to our second cycle.

## V. SECOND CYCLE: "I'M MY OWN HERO"

The focus of this cycle was responding to the first lesson, i.e., augmenting INAH's functionality by integrating references to additional resources alongside hero recommendations. The goal was to empower newcomers to independently access the information they required.

Various assets were incorporated, including internal documentation, projects, courses, previously answered questions, and external resources like YouTube videos and Google results. INAH's architecture was also revamped using Milvus<sup>6</sup> for storage and LangChain<sup>7</sup> for functionalities.

<sup>6</sup><https://milvus.io/>

<sup>7</sup><https://www.langchain.com/>

As shown in Figure 2, INAH now provides the answer to the question and a link to the relevant document. Related questions and heroes can also be accessed. For example, when asked about company holidays, INAH extracts the information from and references the pertinent document.

This version was previewed across different profiles at the firm. Newcomers appreciated being able to consult resources on their own, and related questions were deemed useful to avoid bothering senior colleagues. However, concerns arose about populating the application with authentic data, such as CVs, projects, and internal resources. Team leaders and HR expressed reservations about sharing the company's know-how with external parties due to data privacy and security risks.

Ultimately, the company deemed these risks unacceptable. While LLM potential for onboarding was recognized, deploying models with full, in-house data control was considered necessary. This led to exploring open source LLMs.

**Lesson 3:** Relying on third-party LLMs can pose unacceptable data risks for companies. Maintaining control with in-house deployment is needed.

Hence, the third cycle focuses on analyzing open-source LLMs for onboarding.

## VI. THIRD CYCLE: "PROTECT MY HEROES"

While the best known proprietary LLMs (like GPT-4<sup>8</sup>, Anthropic<sup>9</sup>, Cohere<sup>10</sup> or AI21 Studio<sup>11</sup>) are owned by companies, open source LLMs are freely available. Benefits of open source LLMs (like LLaMA [29], YI-34B<sup>12</sup>, Mistral [30], Bloom [31], and Falcon<sup>13</sup>) include transparency, flexibility, cost savings, customization, and community contributions. Moreover, in various evaluations<sup>14</sup>, open source LLMs have proven their mettle, showcasing their ability to deliver significant results.

Resorting to open source LLMs can be a method for companies to create conversational assistants to provide personalized onboarding guidance and support to new hires, while avoiding the privacy risks associated with proprietary models. While the conversation quality of open source LLMs currently trails proprietary models like GPT-4 and Anthropic, privacy considerations alone ground the use of open source LLMs in this setting, as they allow full control over data and model use.

Hence, our goal in this cycle is to investigate the use of open source LLMs in the context of onboarding processes in software development teams.

Our approach involves:

- 1) Conducting semi-structured interviews with newcomers, team leaders, and HR professionals from other departments to gather insights into current onboarding barriers, needs, and best practices from different perspectives. We

<sup>8</sup><https://openai.com/research/gpt-4>

<sup>9</sup><https://claude.ai/>

<sup>10</sup><https://cohere.com/>

<sup>11</sup><https://studio.ai21.com/>

<sup>12</sup><https://huggingface.co/01-ai/YI-34B-Chat>

<sup>13</sup><https://falconllm.tii.ae/>

<sup>14</sup><https://huggingface.co/spaces/lmsys/chatbot-arena-leaderboard>

- intend to broaden our approach to assess the generalizability of our findings.
- 2) Feeding the open-source LLM on relevant onboarding literature and internal LKS Next data like courses, documentation, and past projects. This ingrains the agent with both external theory and internal domain knowledge, allowing it to leverage broad onboarding expertise along with LKS Next specifics.
  - 3) Enabling iterative refinement through evaluation cycles with users. Initial controlled testing will elicit structured feedback from interviewees on the LLM's capabilities and limitations. The software development branch of LKS Next welcomes from five to seven interns every summer. The effort dedicated to their onboarding and the satisfaction of everyone involved will provide assessments of the agent's utility, compared to the results obtained in the previous years.
  - 4) Finally, analyzing findings across evaluation cycles will elicit transferable design principles for onboarding LLMs. The goal is to extract general guidelines on training, knowledge sources, conversation protocols, and evaluation approaches that can inform LLMs in other organizations.

## VII. RELATED WORK

There is a considerable body of work focusing on onboarding in software development teams. However, to the best of our knowledge, exploring the potential of LLMs in facilitating the onboarding process in software development teams has received little attention so far.

Supportive practices like prompt question answering and mentorship can aid in overcoming the barriers newcomers face [18], [3], [32], LLMs show potential to enhance personalized, instant onboarding support and mitigate productivity losses from training newcomers [5].

In line with our approach, prior work has identified various onboarding facilitation activities based on Bauer's model [8], which outlines a comprehensive framework for effective onboarding practices, including orientation and coaching. Suggested orientation activities include running onboarding sessions to explain the big picture [16]. Within coaching activities, prior work has identified mentoring [10], [11], [12], [13], [14], [15], identifying experts, and answering questions quickly [32]. These coaching actions fall under the broader category of peer support [12]. However, these activities place demands on senior team members providing the assistance. Our approach aims to mitigate their effort by having the LLM provide an initial round of personalized guidance to newcomer questions. The LLM can handle common inquiries, only passing to colleagues those needing more nuanced domain expertise. This reserves peer support for when human experience is most beneficial, while allowing the LLM to resolve routine issues that would otherwise distract experts from their regular work.

Sharma and Stol derived a theoretical model to study onboarding success factors for software developers [9]. Ana-

lyzing various activities, they found that providing continuous support so newcomers feel supported in their job has the greatest impact on onboarding success. This finding aligns with our goal of using an LLM to deliver ongoing, personalized support, as the LLM can be available anytime to answer newcomer questions without taxing expert time. Unlike generalized training, the LLM provides tailored guidance by leveraging company specifics. Thus, an LLM-powered conversational assistant can provide the type of ongoing, context-specific support identified as most crucial for effective onboarding.

While Chandar et al. proposed conversational systems to offer individualized onboarding assistance [33], their work did not focus on implementation factors like data privacy. In contrast, our work explores training open-source LLMs on companies' data to deliver tailored onboarding while avoiding external dependence risks.

Last, Balfroid et al. propose the use of LLMs to generate code tours for onboarding [34]. While we focus on question answering, they pursue the same goal, increase independence of newcomers while requiring less time from senior developers. Both approaches are complementary.

## VIII. CONCLUSIONS

Onboarding remains a pressing concern in software development teams, with newcomers facing different barriers, senior developers overtaxed for training, and managers accepting reduced productivity. LLMs hold the promise to enhance personalized, instant onboarding support; however, relying on proprietary external LLMs poses significant data privacy risks due to the exposure of internal company information.

This work explores LKS Next's experience leveraging LLMs to deliver tailored onboarding assistance to increase newcomer independence. Through an iterative ADR approach, prototypes showcased LLMs' potential but highlighted privacy concerns and the need for in-house deployment. We ultimately aim to develop a tool based on an open source LLM that provides robust, individualized onboarding across the diverse software teams at LKS Next.

Future work includes testing open-source LLMs in a broader organizational setting and investigating issues related to data quality, bias propagation, and the need for human oversight. While our findings suggest that LLMs can be a useful aid, human supervision and contact with newcomers will remain essential for effective onboarding. This research sheds light on the potential benefits and challenges of integrating LLMs into software developer onboarding processes, offering lessons learned from the experience at LKS Next.

## ACKNOWLEDGMENTS

This research is co-supported by contract MCIN/AEI/10.13039/501100011033 and the European Union NextGeneration EU/PRTR under contract PID2021-125438OB-I00. Funding was also received from the University of the Basque Country (UPV/EHU) under the "University-Enterprise-Society" program (US22/17). Additional support is

provided by contract GIU21/037 under the program "Granting Aid to Research Groups at the UPV/EHU (2021)".

## REFERENCES

- [1] A. Begel and B. Simon, "Struggles of new college graduates in their first software development job," in *Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education, SIGCSE 2008, Portland, OR, USA, March 12–15, 2008*. ACM, 2008, pp. 226–230. [Online]. Available: <https://doi.org/10.1145/1352135.1352218>
- [2] L. Pradel, "Quantifying the ramp-up problem in software projects," in *Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering, EASE 2016, Limerick, Ireland, June 01 - 03, 2016*. ACM, 2016, pp. 9:1–9:4. [Online]. Available: <https://doi.org/10.1145/2915970.2915975>
- [3] S. E. Sim and R. C. Holt, "The ramp-up problem in software projects: A case study of how software immigrants naturalize," in *Forging New Links, Proceedings of the 1998 International Conference on Software Engineering, ICSE 98, Kyoto, Japan, April 19–25, 1998*. IEEE Computer Society, 1998, pp. 361–370. [Online]. Available: <https://doi.org/10.1109/ICSE.1998.671389>
- [4] I. Steinmacher, M. A. G. Silva, M. A. Gerosa, and D. F. Redmiles, "A systematic literature review on the barriers faced by newcomers to open source software projects," *Inf. Softw. Technol.*, vol. 59, pp. 67–85, 2015.
- [5] E. Ritz, F. Donisi, E. Elshan, and R. Rietsche, "Artificial socialization? how artificial intelligence applications can shape a new era of employee onboarding practices," vol. 2023-January, 2023, Conference paper, p. 155 – 164.
- [6] M. K. Sein, O. Henfridsson, S. Purao, M. Rossi, and R. Lindgren, "Action design research," *MIS quarterly*, pp. 37–56, 2011.
- [7] A. Galdos, M. Azanza, B. Pérez Lamancha, and I. Paz, "I Need a Hero: ChatGPT al rescate de los desarrolladores recién incorporados," *Sistedes*, 2023. [Online]. Available: <https://hdl.handle.net/11705/JISBD/2023/1818>
- [8] T. N. Bauer, "Onboarding new employees: Maximizing success," *SHRM Foundation*, 2010. [Online]. Available: <https://www.shrm.org/foundation/ourwork/initiatives/resources-from-past-initiatives/Documents/Onboarding/\%20New\%20Employees.pdf>
- [9] G. G. Sharma and K.-J. Stol, "Exploring onboarding success, organizational fit, and turnover intention of software professionals," *Journal of Systems and Software*, vol. 159, p. 110442, 2020.
- [10] S. Balali, I. Steinmacher, U. Annamalai, A. Sarma, and M. A. Gerosa, "Newcomers' barriers... is that all? an analysis of mentors' and newcomers' barriers in oss projects," *Computer Supported Cooperative Work (CSCW)*, vol. 27, pp. 679–714, 2018.
- [11] R. Britto, D. Smite, L.-O. Damm, and J. Börstler, "Evaluating and strategizing the onboarding of software developers in large-scale globally distributed projects," *Journal of Systems and Software*, vol. 169, p. 110699, 2020.
- [12] J. Buchan, S. G. MacDonell, and J. Yang, "Effective team onboarding in agile software development: techniques and goals," *CoRR*, vol. abs/1907.10206, 2019. [Online]. Available: <http://arxiv.org/abs/1907.10206>
- [13] B. Dagenais, H. Ossher, R. K. E. Bellamy, M. P. Robillard, and J. P. de Vries, "Moving into a new software project landscape," in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1*. ACM, 2010, p. 275–284.
- [14] G. Mehrotra and D. M. Berry, "How to benefit from newbies' domain ignorance in software development projects," *Sci. Comput. Program.*, vol. 204, p. 102593, 2021.
- [15] G. Viviani and G. C. Murphy, "Reflections on onboarding practices in mid-sized companies," in *Proceedings of the 12th International Workshop on Cooperative and Human Aspects of Software Engineering, CHASE@ICSE 2019, Montréal, QC, Canada, 27 May 2019*. IEEE / ACM, 2019, pp. 83–84. [Online]. Available: <https://doi.org/10.1109/CHASE.2019.00027>
- [16] A. Ju, H. Sajnani, S. Kelly, and K. Herzig, "A case study of onboarding in software teams: Tasks and strategies," *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pp. 613–623, 2021.
- [17] C. Ostroff and S. W. Kozlowski, "The role of mentoring in the information gathering processes of newcomers during early organizational socialization," *Journal of Vocational behavior*, vol. 42, no. 2, pp. 170–183, 1993.
- [18] F. Fagerholm, A. S. Guinea, J. Münch, and J. Borenstein, "The role of mentoring and project characteristics for onboarding in open source software projects," in *2014 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, ESEM*. ACM, 2014, pp. 55:1–55:10. [Online]. Available: <https://doi.org/10.1145/2652524.2652540>
- [19] Z. Feng, A. Chatterjee, A. Sarma, and I. Ahmed, "A case study of implicit mentoring, its prevalence, and impact in apache," in *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE*. ACM, 2022, pp. 797–809. [Online]. Available: <https://doi.org/10.1145/3540250.3549167>
- [20] H. Naveed, A. U. Khan, S. Qiu, M. Saqib, S. Anwar, M. Usman, N. Akhtar, N. Barnes, and A. Mian, "A comprehensive overview of large language models," 2023. [Online]. Available: <https://arxiv.org/abs/2307.06435>
- [21] O. N. Mohammadreza Farrokhnia, Seyyed Kazem Banihashem and A. Wals, "A swot analysis of chatgpt: Implications for educational practice and research," *Innovations in Education and Teaching International*, vol. 0, no. 0, pp. 1–15, 2023. [Online]. Available: <https://doi.org/10.1080/14703297.2023.2195846>
- [22] E. Kasneci, K. Sessler, S. Küchemann, M. Bannert, D. Dementieva, F. Fischer, U. Gasser, G. Groh, S. Günemann, E. Hüllermeier, S. Krusche, G. Kutyniok, T. Michaeli, C. Nerdel, J. Pfeffer, O. Poquet, M. Sailer, A. Schmidt, T. Seidel, M. Stadler, J. Weller, J. Kuhn, and G. Kasneci, "Chatgpt for good? on opportunities and challenges of large language models for education," *Learning and Individual Differences*, vol. 103, p. 102274, 2023. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S1041608023000195>
- [23] C. K. Lo, "What is the impact of chatgpt on education? a rapid review of the literature," *Education Sciences*, vol. 13, no. 4, 2023. [Online]. Available: <https://www.mdpi.com/2227-7102/13/4/410>
- [24] S. MacNeil, J. Kim, J. Leinonen, P. Denny, S. Bernstein, B. A. Becker, M. Wermelinger, A. Hellas, A. Tran, S. Sarsa, J. Prather, and V. Kumar, "The implications of large language models for CS teachers and students," in *Proceedings of the 54th ACM Technical Symposium on Computer Science Education, Volume 2, SIGCSE 2023, Toronto, ON, Canada, March 15–18, 2023*. ACM, 2023, p. 1255. [Online]. Available: <https://doi.org/10.1145/3545947.3573358>
- [25] R. J. Wieringa, *Design science methodology for information systems and software engineering*. Springer, 2014.
- [26] R. L. Baskerville, "Investigating information systems with action research," *Communications of the association for information systems*, vol. 2, no. 1, p. 19, 1999.
- [27] C. Braga, P. Santos Jr, and M. Barcellos, "Help! i need somebody. a mapping study about expert identification in software development," in *Proceedings of the XXXVII Brazilian Symposium on Software Engineering*, ser. SBES '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 154–163. [Online]. Available: <https://doi.org/10.1145/3613372.3613389>
- [28] M. A. Carey and J.-E. Asbury, *Focus group research*. Routledge, 2016, vol. 9.
- [29] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave, and G. Lample, "Llama: Open and efficient foundation language models," 2023.
- [30] A. Q. Jiang, A. Sablayrolles, A. Mensch, C. Bamford, D. S. Chaplot, D. de las Casas, F. Bressand, G. Lengyel, G. Lample, L. Saulnier, L. R. Lavaud, M.-A. Lachaux, P. Stock, T. L. Scao, T. Lavril, T. Wang, T. Lacroix, and W. E. Sayed, "Mistral 7b," 2023.
- [31] B. Workshop, "Bloom: A 176b-parameter open-access multilingual language model," 2023.
- [32] I. Steinmacher, C. Treude, and M. A. Gerosa, "Let me in: Guidelines for the successful onboarding of newcomers to open source projects," *IEEE Software*, vol. 36, pp. 41–49, 2019.
- [33] P. Chandar, Y. Khazaeni, M. Davis, M. Muller, M. Crasso, Q. V. Liao, N. S. Shami, and W. Geyer, "Leveraging conversational systems to assist new hires during onboarding," in *Human-Computer Interaction - INTERACT 2017*. Cham: Springer International Publishing, 2017, pp. 381–391.
- [34] M. Balfroid, B. Vanderose, and X. Devroey, "Towards llm-generated code tours for onboarding," in *Proceedings of the 2024 ACM/IEEE International Workshop on NL-based Software Engineering (NLBSE '24)*, 2024.

# Mastering Software Craftsmanship: An Industrial Practitioner Perspective

Arastoo Barji

*Siemens AG*

*Digital Industries, Process Automation*

Erlangen, Germany

arastoo.barji@siemens.com

Thomas Jachmann

*Siemens Healthineers AG*

*Diagnostic Imaging, Computed Tomography*

Forchheim, Germany

thomas.jachmann@siemens-healthineers.com

Frances Paulisch

*Siemens Healthineers AG*

*Head of Software Initiative*

Forchheim, Germany

frances.paulisch@siemens-healthineers.com

**Abstract**—Software Craftsmanship is about professionalism in software development. The “craftsmanship” metaphor is based on the notion of professional craftsmen who take responsibility for their work and aim to achieve mastery – not only as individuals but also as part of a community. Especially in the context of developing and extending high-quality systems with long lifecycles, it is important that we nurture a strong culture of professionalism in software development. This includes both technical aspects such as automated tests, addressing technical debt, and refactoring as well as cultural aspects such as responsibility, failure culture, embracing change, and continuous learning. This paper describes what we, as industrial practitioners, mean by professionalism and describes our global training program that sets the foundation for this way of working. This is accompanied by a software craftsmanship program that enables software engineers to continuously learn, further increase their mastery of craftsmanship and apply what they learned on their current and future projects.

**Keywords**—software craftsmanship, software development, software engineering education, code quality, test-driven-design, technical debt, technical excellence, professionalism, learning culture, collaboration

## I. INTRODUCTION

Software is an essential part of today’s world and used across a broad set of application domains. We rely on systems that must have a high quality but also are continuously enhanced. Especially in medical solutions, quality is paramount. Such systems are typically complex and often are “cyber-physical” systems of systems meaning they involve a complex combination of hardware and software. The development of such systems requires not only highly skilled software engineers but also an organization that supports applying good software engineering practices. As industrial practitioners, we have found this combination of individual and organization to be best embodied in the “software craftsmanship” metaphor. As described in [1] software craftsmanship is about: “*Not just the professionalism of the programmer, but also the professionalism of the whole software organization.*”. Software craftsmanship is only one of the topics we address and support with trainings, others are, e.g. in the area of agile, architecture, and testing. We hope that by sharing our experience as industrial practitioners on the important topic of software craftsmanship, that this will enable others to transfer some of the approaches to their own context.

This paper will focus on how we have a two-part approach towards cementing software craftsmanship as a foundation for our organization. Firstly, through a “Software Craftsmanship Training” for individual developers. Secondly, through a “Software Craftsmanship Program” that systematically encourages further mastery of craftsmanship both for the individual as well as the whole organization – and this explicitly applied “on the job” in projects.

Persons driving software engineering education should be aware of the “software craftsmanship” topic, especially how important it is and effective means of learning it. Everyone serious about qualification must focus not only teaching various techniques, but also on how to teach, and especially provide the opportunity to experience, hands-on, the positive impact of a new way of working. The focus for software craftsmanship is explicitly different from many other software engineering training topics also because it targets a change in mindset that leads to the understanding that how you write the code is at least as important as the code itself. Furthermore, such complex systems are developed in teams. Therefore, the collaboration within the team and nurturing a code base that is easily understandable and extendible is important to learn and to experience by applying these approaches on a daily basis. Finally, a culture of continuous learning should be instilled – a way of thinking about problems and a self-reflection that goes far beyond product development.

## II. RELATED WORK

The topic of software craftsmanship has been around since the late 1990s and has recently been gaining increasing popularity. One of the earliest books was “The Pragmatic Programmer: From Journeyman to Master” [2], closely followed by “Software Craftsmanship – The New Imperative” [3]. These laid the foundation for the software craftsmanship movement among software practitioners. One of the most influential persons in this space is Robert C. Martin who published books on “Clean Code” [4,5] and established a common understanding of that term. The books by Martin as well as the more recently published book by Sandro Mancuso “The Software Craftsman: Professionalism, Pragmatism, Pride” [1] are major sources of information on this topic for us. Furthermore, Kevlin Henney’s work in this area, for example [6], had a very strong impact on us and he is one of the key trainers of our Software Craftsmanship Training.

The trend towards increased focus on software craftsmanship is similar to the earlier increased focus on agile. Many are familiar with the agile manifesto (<https://agilemanifesto.org/>). The “Manifesto for Software Craftsmanship”(<https://manifesto.softwarecraftsmanship.org>) builds and extends on the agile manifesto with a focus on technical excellence and an appropriate mindset as well as a focus on community and of helping others to learn the topic.

The craftsmanship community explicitly uses the “craftsmanship” metaphor to focus on high quality, pride, professionalism, and a community. Recall the old baker, shoemaker, carpenter guilds and that each of these were proud experts in their respective fields. They each had communities and explicitly focused on learning the trade in a hands-on manner and accompanying the learning process with mentoring to progress from being an apprentice to a

journeyman to a master craftsman. In their work “Apprenticeship Patterns: Guidance for the Aspiring Software Craftsman” Hoover et al. [7] use the design pattern format to provide many insights and guidance for aspiring software craftsmen.

Teaching software craftsmanship is closely connected to teaching agile software development approaches. For example, in [8] the authors describe teaching software craftsmanship in the context of test-driven-design (TDD) and clean code as part of a course on agile software development. A few years later a closely related author team reported on teaching agile collaboration skills [9]. Again, this was in an academic setting, but the authors particularly pointed out the benefits of clean code and craftsmanship – in particular, they wrote that following and discussing the “Clean Code” guidelines has a positive impact on the quality and design of the resulting code in the projects. In a recent paper focused on interviews with agile practitioners [10] the authors separate “technical excellence” into 4 topics - software craftsmanship, software quality, mindset for excellence, and consistency with good software engineering practices. We do not agree with the separation as to our experience software craftsmanship includes also the quality and mindset aspects. However, we do agree with the importance they place on learning these topics in practice and not only in formal learning. They provide a set of recommendations, e.g., that craftsmanship should be internalized through apprenticeship, that we fully agree with and we have a strong focus on learning from practical hands-on examples and as well as continuously learning “on the job”. The paper [11] had a particularly interesting approach of focusing explicitly on learning in areas that one was weak in and to learn from a mentor how to improve.

### III. MASTERING SOFTWARE CRAFTSMANSHIP

There are not many papers about the learning aspects of software craftsmanship for industrial practitioners. That is one of the novelties of this paper which explicitly focuses on the experience with teaching and otherwise accompanying the learning of software craftsmanship at Siemens and Siemens Healthineers.

For us, mastering software craftsmanship is a necessary foundation to ensure quality in our complex and long-living systems with high quality and regulatory requirements and realized by globally distributed teams. The following two sections of the paper describe the two major parts of these activities aimed at mastering “software craftsmanship”:

- The “Software Craftsmanship Training” was established in order to set the foundation. We have meanwhile trained ca. 300 persons from a broad set of business units over the past 7 years.
- The “Software Craftsmanship Program” which enables software engineers and their teams to progress, at their own pace, up a series of steps, mastering the next levels – all the while applying what they learned on their current projects and reflecting on their learnings in a community of like-minded experts.

### IV. SOFTWARE CRAFTSMANSHIP TRAINING

We have a Software Craftsmanship Training that lays the foundation and common terminology of high-quality software development across an organization. For many participants, this training is a starting point of their journey for mastery in Software Craftsmanship. The goal is to create awareness for

the importance of well-crafted code and that participants intrinsically internalize principles and practices which help them keep control of the code over the entire application lifecycle. The intention of the Software Craftsmanship Training is to generate an insatiable thirst for further knowledge and trainings and is the basis for a mindset change in this regard. The training is designed in a way that it can be performed in a completely virtual environment using just an online whiteboard and a video conferencing application.

This training is not only relevant for beginners or junior developers but can and should also be taken by experienced developers or architects who do not know the different aspects of Software Craftsmanship yet. Advanced training courses for senior developers and architects build upon this training and it is important that the vocabulary and principles of Software Craftsmanship are not only known and lived by developers.

While many agile courses mainly focus on the process of software development, in this training we discuss consequences of poor code as well as design and teach how to achieve technical excellence and agility in writing code which is responsive to change and stays maintainable over time.

As mentioned above, the Software Craftsmanship Training does not only impart knowledge but also strives to strengthen the participant’s mindset towards professionalism in software development. Before focusing on what is meant with professionalism, there is one important requirement for such a training to be mentioned here: whatever we teach in regards to the software development process, methodologies, and technical aspects, it needs to scale and reach a large number of development teams across the company. The transition of a software producing organization in the direction of a quality focused, agile, and highly professional unit of collaborative individuals can only be achieved if a critical mass of adequately trained people is convinced of the necessary measures. The set-up of software development teams is subject to change, just like the requirements of a software application. i.e., it is not only important to react to change in regards of application requirements but also change of team structures including new developers who have to understand, extend, and maintain the code base of an application over the application lifecycle.

To better understand the different blocks of this training, it is important to explain what we mean by professionalism in software development, and what we expect from it. Based on the earlier cited work of Martin [4,5] and Mancuso [1], but focused on our organizations, professionalism in software development for us means:

- **Simplicity** – Not the developers producing the most sophisticated and hardest to understand code are considered the most mature and senior developers, but the ones who are able to achieve the same goals in the simplest possible way. It is vital that other developers are encouraged to actively join in and extend the code with the certainty that they improve it and do not fear adding new issues.
- **Culture of responsibility** – Developers care about code quality and feel responsible to keep it at a high level. At the same time, they are motivated to become better on their personal journey of mastery, like craftsmen.
- **Awareness** – Developers are aware of the consequences of poor code and design and can effectively communicate with a common vocabulary.

- **Failure culture** – A good discussion and failure culture allows teams to learn from each other to improve over time and to adapt existing best practices to their own needs and continuously learn from failures instead of hiding their weaknesses.
- **Technical agility** – Embracing change is not limited to the process but also to the code base. Professional individuals in the software development process know what effort and measures are needed to write code which is responsive to change. Change is not only triggered by new domain requirements but also by the need to reduce unnecessary complexity and technical debt.

Several constraints influence the design of our Software Craftsmanship Training. As customary for education for industrial practitioners, we assume that participants are self-responsible and motivated to learn. Although we have high expectations in the outcome of the training, in regard to the overall transformation of the organization, no participant is forced to learn the content. Nor is there any exam which puts external pressure on the participants to learn. In addition to that, the knowledge level of participants is usually heterogenous which means that some already have a settled opinion about software development which might not always match with software craftsmanship principles. The training aims to initiate a change in the participants' habits and prevent them from falling back into old patterns after they have completed the training.

The training is structured in two two-day workshops followed by a 5th day several months later where the participants jointly reflect on and discuss their learnings. Furthermore, there is a kick-off before the first workshop and homework between the workshops. The number of participants is maximum 16. It has a virtual set-up so that participants from many different locations can come together.

The Software Craftsmanship Training has been designed based on the “Accelerated Learning” principles that include viewing learning as creation and not consumption, recognizing that collaboration aids learning, and that learning comes from doing the work itself [12]. Many aspects of the Software Craftsmanship Training are designed according to the six trumps approach [13] and the 4Cs model from “Training from the Back of the Room” (TBR) [14]. The six trumps are: movement trumps sitting, talking trumps listening, images trumps words, writing trumps reading, shorter trumps longer, and different trumps same. The 4Cs stand for:

- **C1: Connections** – get participants to make connections with what they already know or think they already know about the training topic,
- **C2: Concepts** – provide participants with need-to-know concepts and new information in multisensory ways – e.g., seeing, discussing, writing, reflecting, imagining, and teaching it to others,
- **C3: Concrete practice** – get participants to actively practice the new skills and know-how they have learned,
- **C4: Conclusions** – get participants to summarize what they have learned, evaluate it, and create action plans for how to use the knowledge or skills after the training.

---

<sup>1</sup> SOLID is an acronym formed from the first letters of a set of design principles recommended by Robert C. Martin [4].

The training begins with a kick-off meeting where participants get prepared for the planned workshops. Here the main goal is to motivate them by explaining the purpose of the training and giving them an outlook on the path to mastery in Software Craftsmanship. We use mastery and purpose as motivators [15] to generate an emotional and cognitive impulse. In the kick-off we already create a link to the participants with common problems and “pains” of unmaintainable code and all following consequences. As most of the learners are affected by similar “pains”, they usually can follow our training agenda and our upcoming intentions.

Furthermore, the participants are given homework which is to each choose a different design principle from a pre-defined list. We expect them to get familiar with the chosen design principle and to prepare a short “elevator pitch” (i.e. short presentation) for it. Each learner will explain their design principle to the other participants during the first workshop. Beside this homework, learners get another task to view a video from a list of selected videos about different Software Craftsmanship aspects, like a specific design principle or testing practices. Every learner is asked to write a short summary about the content of the video to point out the main messages. This task is not reflected during the workshop anymore. Its main purpose is to make the learners get the right spirit of the training and already initiate the learning process according to C2 of the 4Cs model [14] by watching, listening, and finally writing a summary of a training relevant topic.

The first workshop starts with a motivational talk by the trainers about principles and qualities of good code followed by the participants presenting the design principles elevator pitches. The opening session is based on the C1 construct of the 4Cs model. Explaining the gained insights to other participants deepens the learners understanding about it. At the same time this is a good opportunity for the participants to get to know each other better.

During the training we do not follow a very strict agenda of the planned sessions. We rather put all training blocks on an online Kanban board so that we are flexible to extend or shrink sessions according to the participants’ needs. This approach gives the trainers and learners room for discussions and clarification of important questions and increases the acceptance of the participants. Our experience shows that group dynamics of every training run vary. Topics which seem to be obvious for participants of one run can lead to intensive discussions in another run. The Kanban-based agenda helps to master the unpredictable progression of a training and is an approach we recommend.

One important learning objective of the training is to teach students how technical agility can be achieved besides process agility. The learning objectives of the training, displayed in Fig. 1 and Fig. 2, can be clustered in three groups. These are:

1. generating know-how and common understanding of technical best practices to achieve a high habitability of the code, such as unit testing and characteristics of good unit tests, the most important principles, related to clean code as well as other design principles e.g. SOLID<sup>1</sup>, Boycout Rule<sup>2</sup>, YAGNI (you aren’t going to need it) etc. as well as the understanding of loose coupling and high cohesion,

---

<sup>2</sup> The Boy Scouts have a rule “Always leave the campground cleaner than you found it” and Robert C. Martin established a similar rule for changing code.

2. teaching methods and paradigms which we also call “tools of craftsmen” such as shift left in testing especially by applying TDD and BDD (behaviour driven design), pair and mob programming, code reviews, katas, coding dojos, etc.
3. uncovering blind spots and creating awareness for the consequences of poor code as well as reasons for unmaintainable code, such as identification of code and design smells, an unreliable test suite, different types of technical debt, insufficient testability of the code, and the importance of simplicity and good naming.

We take YAGNI as one example to demonstrate how a design principle is addressed in the training. This principle sounds very generic and most developers have heard this acronym at some point in their career. This principle is perfect to be combined with the practice of TDD where we teach students to implement something in the simplest way possible. Here usually the discussions of participants start, being convinced that there are sometimes cases where we need to look ahead and implement it in a way that a future scenario is also covered. That's the point where we first of all let the student responsible for this principle explain all the knowledge he/she has collected about it and support it with our position and opinions about home-made complexity and speculative generality. That's the important moment where students identify their own blind spot that some complexity of the code does not just come from the nature of the complex domain or the complex visions of a product owner, but that they themselves sometimes implement things in an overly complicated way which can lead to over-engineering or unnecessary code. Then, repetition of these insights help to support the learning process during all the TDD writing and the review of all groups' TDD implementations. Finally, during the pain code examples, we identify exact places in production code where the students at this point usually notice that this principle is not just something theoretical that only works for simple TDD examples, but that the violation of it really causes huge pain in the project's productive code. A concrete example of YAGNI is when developers start making things configurable when they don't understand a requirement or find boundary conditions. And instead of asking the product owner and clarifying, they either make assumptions or, just to be future proof, they make it configurable in the code. This adds significant complexity to the code which in most of the cases is never really needed again. It remains in the code and needs to be maintained whenever someone touches it.

Taking into account that software for the medical domain is subject to high regulatory and quality requirements as well as the essential complexity of the technical domain, it is crucial to prevent accidental complexity to slowly creep in and cause technical debt and code erosion over time. Thus, one main learning objective of the training is to strive for simplicity. And here the TDD approach is one of the best methods to teach developers to fulfill requirements by implementing them in the simplest possible way and this also automatically increases testability and code coverage. One side effect is that students learn what it means to be the first user of their own code, also known colloquially as “eating your own dog food”.

The second important learning objective is the awareness of how crucial testability and a good test suite are to allow developers to extend and modify the code over its entire lifecycle with trust. Such a “safety net” of good tests enables

technical agility and responsiveness to change and increases the confidence to perform refactoring whenever needed.

The three mentioned clusters of the training cannot be conveyed with pure lecture-style presentations. Here we use techniques to accelerate the learning process like the 4Cs of TBR mentioned previously. Whenever possible, we arrange sessions where students need to explain what they learned individually or in a group discussion to the other participants.

Another successful practice of the training are hands-on TDD coding dojos which we preferably perform at cyber-dojo.org. We do simple TDD examples in pairs to teach the principles of this development method and take enough time afterwards to review the code of every single group in a plenary session so that everyone can learn from the experienced trainers. These reviews are very valuable to also address aspects like naming, testability, design principles etc. along the way.

Day 1	Day 2
Welcome	Recap of Day 1
<b>Introduction + Motivation</b> <ul style="list-style-type: none"> <li>- Why principles matter</li> <li>- Qualities of good code</li> </ul>	<b>Clean Code</b> <ul style="list-style-type: none"> <li>- Good/well crafted code</li> <li>- Bad examples and solutions proposal</li> </ul>
Elevator pitches (homework)	<b>From Clean Code to principles</b> <ul style="list-style-type: none"> <li>- The most important principles</li> <li>- Bad examples and solutions proposal</li> </ul>
<b>TDD / BDD introduction</b> <ul style="list-style-type: none"> <li>- The importance of BDD to product code, serve as specification, ensure agility</li> </ul>	<b>Craftsmanship</b> <ul style="list-style-type: none"> <li>- Motivation / Awareness</li> <li>- From novice to expert</li> <li>- Katas</li> <li>- PDSA cycle</li> <li>- UX, PX and habitability</li> <li>- Code reviews and communication</li> <li>- Rule of three</li> </ul>
<b>Discussion</b> <ul style="list-style-type: none"> <li>- How can we apply BDD in daily work?</li> <li>- How to deal with existing code bases</li> </ul>	<b>Wrap-up of Session 1</b> <ul style="list-style-type: none"> <li>- Outroduction</li> <li>- Homework</li> </ul>

Fig. 1. Structure of the training blocks of workshop 1

Day 1	Day 2
<b>Introduction of the Clean Code Cheat Sheet (CCCS)</b> <ul style="list-style-type: none"> <li>- Rewrite and refactor</li> <li>- Techniques and best practices</li> </ul>	<b>Handle Legacy Code</b> <ul style="list-style-type: none"> <li>- Rewrite and refactor</li> <li>- Techniques and best practices</li> </ul>
<b>CCCS game</b> <ul style="list-style-type: none"> <li>- Identify unknown rules</li> <li>- Explain unknown rules</li> </ul>	<b>Technical Debt</b> <ul style="list-style-type: none"> <li>- Quadrant</li> <li>- Exercises (examples)</li> </ul>
<b>Discussion</b> <ul style="list-style-type: none"> <li>- What makes you a professional developer?</li> </ul>	<b>Pain Code Walkthrough</b> <ul style="list-style-type: none"> <li>- Go through most painful code</li> <li>- Find solutions and discuss how to avoid in the future</li> </ul>
<b>Code immersion cont'd</b> <ul style="list-style-type: none"> <li>- Bringing together the topics from the previous 3 days and put them into practice (coding kata)</li> <li>- TDD / BDD incl. examples and exercises</li> </ul>	<b>Final Discussion:</b> <ul style="list-style-type: none"> <li>- Revisit „What makes you a professional developer“</li> <li>- What will I change tomorrow</li> <li>- How can I bring this into the organization</li> <li>- Define concrete action items per participant and follow-up meeting</li> </ul>

Fig.2. Structure of the training blocks of workshop 2

Often, students fall back to their old habits a short time after a training, or some of them believe that most things they learned cannot be applied to their complex world. In order to address these problems, we give them homework between workshop 1 and 2 which is called “pain code walkthrough”. Here they are asked to go through their real project code and after 4 – 6 weeks to come back to workshop 2 with some piece of code which causes pain. Participants need to apply everything they learned in workshop 1 to real world code of their projects and are often astonished about the new insights they get about it. During the group review of their pain code examples in workshop 2 students recognize that many projects suffer from similar problems. This increases their common understanding and vocabulary of well-crafted code as well as learning indicators for pain code and code smell.

There are also additional support materials available – e.g., templates for discussion, online videos on particular topics, collections of best-practices from previous runs of the programs, etc. These are used to further support the use in the projects. The online videos are helpful for making the abstract topic easy to understand for non-software persons – for example a video based on [16], about how Software Craftsmanship is like gardening, i.e. you must continuously “maintain” your garden for it to be healthy and productive.

**Benefits of the Training:** We have initial insights about the positive impact of the training program gathered through a qualitative questionnaire distributed after the training. As shown in Fig. 3, the participant’s evaluation of the importance of testability before the training compared to after the training is significantly improved. Furthermore, 45% of the participants reported that the training had an impact beyond their own responsibility i.e. on their team or organization. Reported changes in behavior included being far more expressive (method names, variable names, etc.) in the code, writing code so that it can be easily understood and maintained by coworkers, and more emphasis on testing as well as following “Clean Code” guidelines. 56% viewed the content of the training to be extremely important (5 on a scale of 1 to 5). 61% viewed the training to be either somewhat or extremely impactful (4 or 5 on a scale of 1 to 5). Also the mix of theory and hands-on work was reported to be valuable.

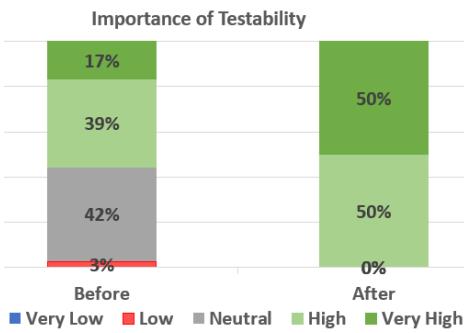


Fig.3. Participants evaluation of the importance of testability

## V. SOFTWARE CRAFTSMANSHIP PROGRAM

In order to build on this and to achieve sustainable competency growth in the area of software craftsmanship, we need to provide an environment which is open to such activities and enables the developers to take the time to work on such improvements in their way of working rather than solely focusing on “working features”. We have thus established a “Software Craftsmanship Program” within the organization to have a systematic way to learn in a self-motivated way, but not only as individuals also as teams. Recall, the team aspect is particularly important, especially to have a common understanding across the team which is easier to understand and enhance.

It is no secret that a team will work most efficiently when the individual team members share the same values. Therefore, the question: “What do we value?” is something the team (and the whole organization) needs to discuss repeatedly. Shared values are required for people to be able to commit to the same resulting principles. Each of these principles will be turned into a set of practices which will guide everyone’s daily work and carry the team values. While we have a common approach for Software Craftsmanship Programs, there are variants in different business areas of our organization due partly to business specific differences.

However, the programs have more similarities than differences, and the common core will be described here.

The core idea is that high-quality code is best written by professional and motivated developers and by the understanding of the organizations that this requires an investment into continuous learning of their experts as well as an environment that supports their learning. The learnings are designed around core principles:

- Learnings of each software developer
- Learnings of the whole team
- Understanding and writing code
- Understanding and writing tests / test code
- Room for topics that are selected by the team to allow them also to focus on their own desired learnings.

Within Software Craftsmanship it is of high importance to constantly discuss, evolve, and document these values, principles and practices – the so-called “team standard”. There is a very strong focus on mindset and the involved teams explicitly are asked to take the time from their busy schedules to work on discussing these topics and agreeing on how to work together in their teams. Management funds and provides time for the teams to work on their software craftsmanship learnings on a weekly basis for experience exchanges.

Gamification is a desired supportive aspect. As a low-effort way to show progress, yet also add a bit of gamification, there are a set of levels – so that a team gradually moves up the ladder as they get more experience with applying this approach of self-learning (see Fig. 4). Some organizations name their levels after martial-art belts, other after the planets of our solar system. Some even utilize a bit of a friendly competition between teams to compete who reaches the next level first, leading at the end to a team success which is typically celebrated by all teams. The “Clean Code” [4] approach is a common thread through many of the topics e.g. on one of the lower levels, having discussions about chapters of the “Clean Code” book and at other layers discussing how to do “error handling” according to “Clean Code” guidelines. Gamification has had a positive effect of encouraging and visualizing progress but at developer’s own pace.

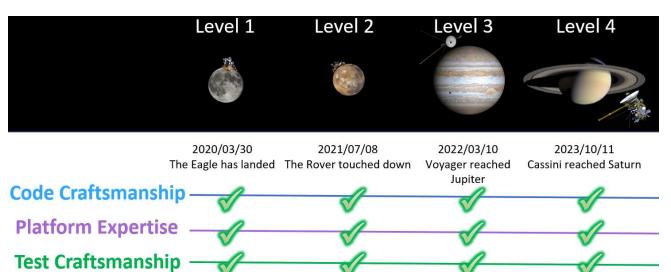


Fig. 4. Example of a gamification system illustrating a journey through Software Craftsmanship levels from a business unit showing that multiple groups have reached level 4

By working on the different Software Craftsmanship levels, where each level provides new stimulus to be discussed within the teams, the participants discover new and old things which are worth to discuss in the team. Many might eventually become a part of the team standard principles and practices.

Another common aspect of the program is a focus on “Test Craftsmanship” where test-driven design and agile testing approaches plays a key role. Here a focus is on improving the focus of the various levels of testing, improving testability

(both analyzability and observability) of the product, as well as setting the foundation to reach a high level of maintainable automated testing while ensuring fast feedback through a balanced test pyramid.

The programs have shown that they sustainably reinforce the craftsmanship mindset in the teams' daily work routine. For example, that developers have a common mindset across the team and think harder about maintainability and potential code sharing e.g. by selecting design/naming decisions that support simplicity, and allowing others to join in. We encourage software craft "experts" to mentor more junior colleagues. For example, with our coding dojo moderator training where participants practice moderating each other, they subsequently moderated other team members who were completely new to the topic. This helps enable a regular focus on applying software craftsmanship approaches.

We do see positive impacts during our focus on software craftsmanship, but it is difficult to make a precise correlation to what extent the reduction in defects that we see is due to software craftsmanship as there are also other improvements that are done and many of these topics are interrelated.

## VI. SUMMARY AND OUTLOOK

Our learning activities for software craftsmanship are embedded in a culture of modern software engineering that provides fundamental cornerstones that align well to the software craftsmanship topics. These include:

- Creating well-structured maintainable automated tests must be a non-debatable part of code development. They must form a test pyramid that has a clear eye on fast feedback cycles.
- There must be means in the project to deal with encountered technical debt, i.e. corridors that can be planned by the architects to deal with encountered drift through changing environments or changing requirements.
- Refactoring in the small must be possible where code changes identify problems (e.g. due to new requirements)
- A "done-done" culture must be fostered, where finishing a topic also includes all means of stabilization (compiler warnings, static code analysis results, etc. are considered as well as defects identified by automated / manual tests)
- It requires a build and test environment that is running "green-to-green", i.e. test flakiness is seldom and developers can trust their infrastructure.

We have found that a focus on software craftsmanship is a very effective way to instill this mindset to have high-quality, maintainable, testable code written according to common team standards. As a consequence, we experience that it is much easier for a different person to understand and change the code. There are inherent close links to the testing and architecture roles and associated activities e.g. test-driven design and reducing technical debt. As Sandro Mancuso wrote in [1], "*Software craftsmanship is a long journey to mastery. It's a mindset where software developers choose to be responsible for their own careers, constantly learning new tools and techniques and constantly bettering themselves.*"

The craftsmanship training is in the process of being aligned with other training endeavors in our companies. For example, strengthening the alignment between software developers who are craftsmen, and our other training

programs such as those on software architecture and testing. Furthermore, we are considering the extent of commonality and variability across the software craftsmanship programs in the different business units to decide how to align further across the company. We also plan to make software craftsmanship an essential skill of a new "lead developer" role and this role has a high responsibility in our software engineering roles.

We hope that our key insights, summarized below, enable other organizations to master software craftsmanship and encourage more focus on software craftsmanship in software engineering education and training.

- Craftsmanship is an important, yet perhaps still underrepresented software engineering education topic.
- How to teach craftsmanship through a mix of presentations and hands-on coding with real examples by trainers who are expert craftsmen.
- Addressing the mindset is challenging but especially important. As software-driven products become more prevalent and mission-critical, the need for software engineers who are not only highly-skilled technically, but also have a mindset that values professionalism, high-quality, and continuous learning will grow.
- In addition to one-time training, consider to establish a "program" or community for further continuous learning.

## REFERENCES

- [1] S. Mancuso, *The Software Craftsman: Professionalism, Pragmatism, Pride*, Pearson, 2014
- [2] A. Hunt, D. Thomas, *The Pragmatic Programmer: From Journeyman to Master*, Addison-Wesley Professional, 1999
- [3] P. Breen, *Software Craftsmanship – The New Imperative*, Addison-Wesley Professional, 2001
- [4] R. C. Martin, *Clean Code: A Handbook of Agile Software Craftsmanship*, Pearson Education, 2009
- [5] R. C. Martin. *The Clean Coder: A Code of Conduct for Professional Programmers* (1st. ed.), Prentice Hall Press, 2011
- [6] K. Henney, *97 Things Every Programmer Should Know: Collective Wisdom from the Experts*, O'Reilly, 2010
- [7] D. Hoover and A. Oshineye, *Apprenticeship Patterns: Guidance for the Aspiring Software Craftsman*, O'Reilly, 2010
- [8] M. Kropp and A. Meier, "Teaching Agile Software Development at University Level: Values, Management, and Craftsmanship", 2013 26th International Conference on Software Engineering Education and Training (CSEE&T). IEEE, 2013
- [9] M. Kropp, A. Meier, and R. Biddle, "Teaching Agile Collaboration Skills in the Classroom", 2016 IEEE 29th International Conference on Software Engineering Education and Training (CSEE&T). IEEE, 2016.
- [10] A. Alami and M. Paasivaara, "How Do Agile Practitioners Interpret and Foster "Technical Excellence"?", Evaluation and Assessment in Software Engineering (EASE 2021), 2021
- [11] T. Sedano, "Towards Teaching Software Craftsmanship", 2012 IEEE 25th International Conference on Software Engineering Education and Training (CSEE&T). IEEE, 2012
- [12] D. Meier, *Accelerated Learning Handbook*, McGraw-Hill, 2000
- [13] S. L. Bowman, *Using Brain Science to Make Training Stick*, Bowperson Publishing and Training, Inc., 2010
- [14] S. L. Bowman, *Training from the Back of the Room: 65 Ways to Step Aside and Let Them Learn*, Pfeiffer, 2008
- [15] D. H. Pink, *DRIVE: The Surprising Truth About What Motivates Us*, Penguin Publishing Group, 2011
- [16] B. Venners, "Programming is Gardening, not Engineering", 2003, <https://www.artima.com/articles/programming-is-gardening-not-engineering>

# Experience Report of the AWS+KCL Impact Accelerator for Public Sector Engagement

Caitlin Bentley  
Department of Informatics  
King's College London  
London, UK  
[caitlin.bentley@kcl.ac.uk](mailto:caitlin.bentley@kcl.ac.uk)

Daisy Ogden  
EMEA SARTL  
Amazon Web Services  
London, UK  
[ogddaisy@amazon.co.uk](mailto:ogddaisy@amazon.co.uk)

Joanna Walker  
Department of Informatics  
King's College London  
London, UK  
[johanna.walker@kcl.ac.uk](mailto:johanna.walker@kcl.ac.uk)

Elena Simperl  
Department of Informatics  
King's College London  
London, UK  
[elena.simpperl@kcl.ac.uk](mailto:elena.simpperl@kcl.ac.uk)

Stefanos Leonards  
Department of Informatics  
King's College London  
London, UK  
[stefanos.leonards@kcl.ac.uk](mailto:stefanos.leonards@kcl.ac.uk)

Wenjie Yin  
Department of Informatics  
King's College London  
London, UK  
[wenjie.yin@kcl.ac.uk](mailto:wenjie.yin@kcl.ac.uk)

Mike Bainbridge  
EMEA SARTL  
Amazon Web Services  
London, UK  
[mbbainbr@amazon.co.uk](mailto:mbbainbr@amazon.co.uk)

Gunel Jahangirova  
Department of Informatics  
King's College London  
London, UK  
[gunel.jahangirova@kcl.ac.uk](mailto:gunel.jahangirova@kcl.ac.uk)

Christopher Hampson  
Department of Informatics  
King's College London  
London, UK  
[christopher.hampson@kcl.ac.uk](mailto:christopher.hampson@kcl.ac.uk)

**Abstract**— This industry experience report chronicles the experience of developing an impact-focused group project module within a computer science master's programme at King's College London over two years. The module was set up in collaboration with Amazon Web Services to match student teams with public sector challenges requiring innovative technological solutions. An iterative process of modifications based on partner and student feedback aimed to enhance the learning experience and outcomes. Key benefits included providing authentic professional development for students, enabling innovation and entrepreneurship, building partnerships between academia and the public sector, and embedding responsible innovation into projects. However, challenges emerged around managing expectations, ensuring consistent partner engagement, providing support for spin-outs, and handling sensitive data issues. As more projects involved artificial intelligence applications in the second year, developing mechanisms to ethically provide access while protecting sensitive information was an increasingly crucial need. Moreover, understanding the value and impact of this model of software engineering project module requires additional research support. Overall, this collaborative module offers a promising model to deliver impact-driven solutions through coordinating academia, industry, and public sector partners. Further research can help optimise such partnerships for societal impact.

**Keywords**—group projects, public sector innovation, software engineering education, responsible research and innovation

## I. INTRODUCTION

Public sector organisations in the UK encompass a diverse range of government entities, local authorities, civil society organisations, and other institutions dedicated to serving the public interest. Despite this diversity, a common challenge is the limited availability of resources and in-house expertise for the development of innovative technological solutions to address their complex issues. Often constrained by budgetary limitations and access to technological infrastructure, public sector organisations may face difficulties in keeping pace with

rapidly evolving technologies and experimenting with them to enhance the efficiency, responsiveness, and quality of their services for citizens or stakeholders. Consequently, there is a gap between the potential for technological innovation and the capacity of many public sector entities to take advantage of technological advancements effectively. Moreover, artificial intelligence applications, and the sustainability of technology-enabled services magnify concerns surrounding public sector accountability, necessitating a responsible approach.

Simultaneously, a core component of undergraduate and postgraduate degree programmes in computer science is to support students to engage with real-world challenges, to respond to actual customers, and to gain authentic professional experience. Firsthand interaction provides a practical understanding of customer needs, preferences, and the complexity of addressing authentic challenges. Moreover, students benefit significantly from collaborating within teams to deliver authentic solutions. Teamwork not only fosters the development of interpersonal skills but also mirrors the collaborative environments found in professional settings. Through such experiences, students not only enhance their problem-solving abilities but also cultivate a holistic skill set that goes beyond theoretical knowledge, or simplified problems, and seeks to better prepare them for the workforce.

While it may seem promising to assign student groups to tackle public sector challenges, several issues can arise in the process. One concern is the possibility of students not delivering high-quality work, possibly due to a lack of experience, understanding, or difficulty working in teams. At the same time, it can be difficult for academic institutions to form partnerships with public sector organisations, and they may also face challenges in allocating sufficient time or providing the necessary context and expertise to aid students in understanding the complexities of the issues at hand.

In this report, we chronicle our experience over the past two years setting up, and improving, a year-long group project module within a computer science master programme at a UK

university. The impact accelerator was set up in collaboration with a technology company that already had access to a clientele within the public sector. Additionally, utilising their technology stack and enabling insights for our students into their innovation approach, we collaboratively formulated a training programme and project structure aimed at prioritising impact. The next sections outline prior related work, and our experiences in years 1 and 2 respectively. We end this report with an evaluation of the programme.

## II. RELATED WORK

### A. Group Projects in Computer Science Curriculum

Group projects in computer science curriculum are widely recognised for their potential benefits, such as, fostering collaboration, communication, and problem-solving skills among students [1-3]. However, challenges such as free-riding, unequal participation, and conflict resolution are frequently experienced such that [1] emphasises the importance of aligning project assessment with collaboration-oriented learning objectives and providing adequate support to students. Likewise, related work also focuses on how best to structure group projects to help instructors with practical issues around group formation, role assignment, and setting assessment criteria [4]. This research highlights the importance of having effective scaffolds and support mechanisms in place to maximise the educational benefits of group projects within computer science curricula, emblematic of a student-centred approach and pedagogy. Yet, when introducing industry engagement into the curriculum, there may be a challenge to align the industry needs with diverse student skill development needs. Balancing these aspects requires thoughtful consideration and attention, so that ideally, industry involvement enhances the student learning experience whilst accommodating skill learning objectives and development. Our module is also at a post-graduate level, such that we should expect students to have learnt many of the collaboration-oriented and project management knowledge and skills in previous years, and that this module should enable greater responsibility for applying prior knowledge, although this is not always the case.

### B. Organising Group Projects with Industry Partners

Challenge-based learning is a learning framework first advocated by [5], where students engage in experiential learning through working on open-ended challenges [6]. The instructors, the stakeholders, and the students are to collaborate and on the final product as well as a self-reflective and critical development process [7, 8]. A relevant practice in undergraduate and postgraduate education is capstone projects, which typically happens at the end of a US undergraduate course, emphasising integrating and applying prior knowledge on a client-sponsored problem [9].

The role of industry mentors can range from technical consultants for their approaches and facilities to being part of the project team – sometimes their responsibilities may be seen as not well-defined enough [10]. However, there is limited reflection on what the value or challenges of working with industry partners as module organisers and partners in delivering the programme. There could be many benefits related to streamlining client relations, accelerating innovation, and enabling technology and public sector partners to identify and foster top talent.

### C. Impact-focused Software Engineering Education

In existing challenge-based learning and capstone courses, the clients and knowledge exchange partners tend to be companies, which, compared to the public sector, are much more experienced in software engineering project management. Framing the projects is often the most challenging aspect of organising client-sponsored group projects [11, 12], where the course instructors are often required to help project sponsors define objectives which are challenging, well-defined, but still allow open-ended enquiries [11]. Additionally, it is tricky to ensure that work is proportionately organised across different groups [12].

Specifically for the public sector, [13] argues that digital transformation is "the process of integrating digital technologies and solutions into all aspects of the activities of an organization, whether public or private, implying deep changes in organization and management to account for regulations and a citizen-centric methodology," and that computer scientists are crucial drivers of technological change. Digital transformation is considered important because it enables public administrations to be more responsive to citizens' needs and provide higher quality services [13]. Yet, within the discourse of digital transformation, there is a positive bias towards technological integration. Given pressing global concerns surrounding environmental sustainability, privacy and surveillance of citizens, we must also educate computer scientists to know when technology should not be used within the public sector. Although there has been some discussion regarding applying responsible research and innovation and sustainability competencies in engineering and computer science education [14-16], studies have not yet gone into depth on the specifics of responsible research and innovation (RRI) in a software engineering education context. We adopt Stilgoe et al.'s [17] definition of RRI which "means taking care of the future through collective stewardship of science and innovation in the present." This industry report therefore provides a narrative of our experience of experimenting with different approaches to integrating RRI in the group projects in the run up to more in-depth research which will be conducted in the coming year.

## III. PROJECT FRAMEWORK & OUTCOMES IN YEAR 1

### A. Description of the module design

The module is a year-long research and practice led course in which participants constructed high quality solutions to challenges faced by public sector partners. In the first semester, students critically explored the ways advanced technological solutions are planned, designed, built and governed, as well as how to evaluate the impact their solutions have on public sector performance, issues and communities through lectures and practical workshops. In the second semester, students worked on challenges presented by partners to deliver a substantial research project in teams of 5 or 6. There were minimal constraints on the type of solution teams could create, other than the budget each team was allocated. However, all students enrolled on this course have completed the undergraduate programme at our institution, so we were fully aware of the wide range of computer science modules students are required to complete in their undergraduate degree, such that students could build on pre-existing knowledge to create various software, data-driven services, artificial intelligence applications or otherwise. Teams were

also guided by faculty and technology company partner mentors.

The project was split into three distinct phases: 1) project initiation and design brief; 2) research and development; and 3) iterate and evaluate. Although the project framework and teaching style adopts an agile approach, each team was responsible for developing its own project management and software development approach. Therefore, some teams chose alternative approaches, but still benefitted from the overarching project framework structure emphasising feedback and iteration. Throughout the project, teams had scheduled 15-minute meetings with public sector partners, technology company partner, and faculty advisor separately. Additional practical lab sessions focused on peer learning and feedback were also held each week. Teams were responsible for developing project management plans and implementing them, as well as directing and facilitating meetings. Students were required to complete online tutorials every fortnight focusing on professional skills refreshers and individual reflective activities.

The assessment included five components: 1) engagement in online tutorials; 2) a prototype demo submission and video presentation; 3) a research and critical evaluation report on the project process and outcomes; 4) an individual reflection on project contributions, skill development, and teamwork; and 5) completion of all online tutorials.

#### *B. Group formation*

Our institution values diversity, so we have a policy ensuring that student groups are culturally diverse. Additionally, because we have fewer women and gender non-conforming students enrolled, we require each group to have at least two self-identifying women. This policy aims to create a more inclusive learning environment. However, at the start of the programme, there are usually students that have dropped the course, or likewise, some that arrive late due to student visa issues, which makes the policy difficult to implement until the final enrollment deadline has passed. Moreover, to minimise potential conflicts, and maximise the chance that students work in teams that are composed of individuals with complementary interests, we assign a small group (of 3-4 students) coursework which mirrors a small design task of their project. At the end of the coursework, we collected feedback on the teamwork, their preferred team role, and area of interest. We then combined teams that were working well, and made small changes to address any conflicts or groups in which roles and interests were not sufficiently balanced.

#### *C. Public sector challenges*

Challenges were solicited by AWS, and included various issues that public sector organisations faced surrounding organisational efficiency, accessibility, and sustainability in public services. Teams worked on automating service information updates, or occupational health assessments, streamlining waste collection, optimising employee rostering, visualising data, and improving recycling and green waste services. Projects aimed to provide innovative solutions to the challenges presented, and there was scope to interpret each challenge in multiple ways.

#### *D. Project milestones & assessment*

Given we were flexible with the nature of the projects, and did not stipulate a specific project management approach for

the teams to follow, we established specific project milestones. These milestones served as guiding points, enabling teams to incrementally advance in their work, stay on track, and proactively collect feedback for integration from partners, advisors and peers. There were three core milestones. The first was a design brief presentation in which teams presented their research on the challenge and proposed solutions. All of the student teams, partners and advisors were in attendance, and provided feedback via a short survey. Teams also submitted their design brief for written formative feedback. The second milestone was a fast prototype demonstration. These demonstrations were attended by only the student team's partner and other teams. Lastly, the third milestone was the final demo day. Teams prepared a 15 minute pitch and demonstration of their final solution, which groups of partners, faculty and esteemed guests observed and provided feedback via the same short survey.

#### *E. Partner & student feedback*

Emphasising a feedback-driven process, incorporating weekly meetings, combined with the project milestones provided multiple opportunities for ongoing evaluation and refinement, fostering a collaborative and iterative project development cycle that allowed for continuous improvement and adjustment as needed. We also conducted a feedback session with students, which surfaced a lack of engagement with lecture content, difficulty applying concepts and techniques related to the impact stream of project work, as well as requesting more hands-on practical learning reflective of what they would encounter in their project work. Some issues in how different teams had varying support from partners, and a preference for challenges that had clear public benefit also surfaced. For example, one challenge related to developing an employee rostering platform for seasonal workers, but this was viewed as indirectly related to public benefit and was less valued by students.

#### *F. Project outcomes*

All project teams successfully delivered solutions that met the challenges outlined by the partners. During the final demo day, teams received positive evaluations, with every team surpassing a rating of 3 out of 5 when partners were asked if they would purchase the presented product or service. Remarkably, half of the teams, totaling four out of eight, received ratings exceeding 4 out of 5. Notably, one team developed a spin-out from their solution, attracting interest from multiple partners. Additionally, two other projects garnered interest for further development. Feedback from the technology company partner was also received, suggesting that combining certain projects could create an effective platform. However, an identified weakness was that by the end of the year, most students had already made post-graduation plans or accepted job offers, limiting considerations for advancing their solutions. This module design flaw is addressed in the next section for improvement.

#### *G. RRI integration*

In the first year, we held a series of lectures before the start of the projects surrounding defining impact, technology and AI for social good, fairness, accountability, transparency, ethics and sustainability, as well as case studies on public sector issues. Once the project started, we asked teams to identify which impacts they anticipated in their design brief, and then later in the last section of the project, we gave them activities and scaffolds to help them research RRI methods to

to evaluate the impact of their solution as they defined it. This approach did not sufficiently help students to integrate RRI practise into their project design and development process, as it was disconnected from their planning and implementation work. We also found that teams gave surface level descriptions of impact, and did not plan adequate time to conduct the necessary research into investigating potential impacts.

#### IV. CHANGES & REFLECTIONS IN YEAR 2

##### A. Changes to module design to enhance RRI integration

From the student feedback collected in year 1, we addressed the difficulty students had with connecting what they learned in the first semester surrounding RRI to their project work, especially when applying public sector knowledge and design thinking for impactful solutions. To make things clearer, instead of using lectures, we switched to a flipped learning approach. We found this gave students more time to practice what they were learning, so that when they started their projects, they had prior experience incorporating responsible innovation techniques. Although our curriculum often covers social, legal, and ethical professional issues, students found that it was still challenging to integrate this knowledge into project planning and solution design. We also added a hands-on session to go through project planning, letting students compare and contrast different approaches in a more hands-on way. These changes helped students link their learning with their project work more effectively. We stipulated within the design brief submission that teams needed to indicate how they were implementing at least two RRI methods into their project plan. We added an additional session at the project midway point in which teams used RRI cards to reflect on the effectiveness of their plans. This seemed to spur many of the teams to increase the stakeholder engagement, and in contrast to the previous year, in which only one team conducted a user study within the scope of their project, in the second year, five out of eight teams arranged focus groups and interviews with targeted stakeholders and users.

For the second year, AWS was also able to gather challenges from a more cohesive set of public sector partners, such that we could focus our RRI training. Additionally, in the first year, we scheduled meetings in blocks with partners to be able to monitor them and to ensure smooth collaboration. However, in the current year, we entrusted them to independently schedule and conduct these meetings. This adjustment not only demonstrated increased confidence in the students' ability to manage such interactions but also facilitated partner attendance. By empowering the student teams to take the lead in organising meetings, we aimed to address challenges from the previous year where partners occasionally missed scheduled meetings.

##### B. Entrepreneurship & Changes to Group Formation

To proactively address the potential for spinning out solutions and to encourage students to consider entrepreneurial paths earlier in the process, we incorporated a practical session during the first semester. This session focused on key aspects such as entrepreneurship, licensing, and spin-outs. During the group formation process, we took an extra step by explicitly asking students if they were interested in creating a spin-out. We then grouped students based on their preferences. This approach ensured that students not inclined towards creating a spin-out were more

likely to adopt an open-source license for their solutions. Likewise, students expressing an interest in establishing a spin-out were grouped together, facilitating collaboration and support for those inclined to take their innovative ideas further. This addition aimed to foster a more tailored and forward-thinking approach to the potential entrepreneurial aspects of their projects. We also emphasised that teams can adapt their deliverables and project report to support spin-out plans, but have not seen evidence of students taking up this offer yet as we are just halfway through the second year.

#### V. EVALUATION

Overall, the development of this impact-focused group project module demonstrates promising outcomes in enabling students to apply their technical and professional skills to address complex public sector challenges. The iterative refinements made between the first and second year based on partner and student feedback helped enhance the learning experience and outcomes. We learned a great deal regarding how to integrate RRI into design thinking, and software engineering teaching as well.

Providing authentic real-world experience for students to engage with public sector partners and improve their professional skills was a key benefit that emerged. Feedback indicated that students valued the practical application and professional development. Additionally, the module enabled students to take their solutions further through spin-outs, licensing, or continued interest from partners by fostering innovation and entrepreneurship. Explicitly incorporating entrepreneurial elements into the module in year two was beneficial in this regard. Embedding responsible innovation into projects by making adjustments to the curriculum and project structure also helped improve students' abilities to consider social, legal, and ethical implications. Enabling continuous improvement through ongoing feedback from students, faculty, and partners provided multiple touchpoints for evaluation that allowed incremental enhancements to the module design and learning process.

Furthermore, an emergent concern in year 2 relates to the sharing of sensitive data, as challenges arose for some groups in accessing data from partners that were initially promised in some cases. This year, the majority of projects involve developing artificial intelligence applications or otherwise data-driven services, which require significant amounts of quality data to train and test models in contrast to last year, in which only two projects implemented artificial intelligence. Public sector organisations often process datasets containing personal information that cannot readily be shared with external groups due to the UK GDPR. This poses difficulties for student groups to obtain the data access needed to successfully build and validate data-driven solutions within the constrained timeframe of a one-year module. While generating synthetic datasets can provide an alternative, high-quality synthetic data that preserves underlying patterns is difficult to produce. Developing appropriate data sharing agreements, protocols, and access between public sector partners and academic institutions can enable the ethical and compliant sharing of data for project work, but negotiating these arrangements often requires longer timeframes than this module allows. As artificial intelligence and data-centric solutions continue growing in popularity, strengthening long-term relationships and data governance policies between partners will be crucial to provide the data access students need while protecting sensitive information and adhering to

responsible innovation practices. This is one of the reasons why the role of the technology company in maintaining client relations with the public sector partners could be a crucial success factor of the programme. Moreover, we had hoped that academics within the department would also begin to form partnerships with public sector partners but this has not transpired yet. In the meantime, we may wish to develop further guidance for students on developing synthetic data.

Additionally, some other areas for improvement were identified. Ensuring consistent partner engagement across groups was noted, as some variation in partner involvement occurred. Managing student expectations around project scope and partners' needs is also needed, as some students expressed uncertainties about meeting partner expectations. We noticed, for instance, also that when teams in year 1 focused almost too much on the partners' needs, it often limited the solution potential. At times, teams needed to step back and examine the challenge using a broader lens beyond just the specific feedback from partners, in order to envision solutions with a degree of novelty and impact that went beyond customised fixes for individual partner contexts.

Likewise, ensuring partners – both AWS and the public sector partners – receive meaningful benefits from their time investment, monetary and in-kind contributions in the module is an area requiring further attention. Partners invest substantial time, and it was not always evident what direct returns they achieved, especially because of the students' weaknesses research and reporting on potential impacts. In year 2 we added a requirement to deliver a design package to the partners. Moreover, we also observed benefits that we had not anticipated by hosting all teams and partners in sharing progress during the project milestone events and at the final demo day. These milestones enabled partners to see a breadth of approaches and solutions highlighting promising directions they may not have envisioned. This cross-pollination of ideas among partners was a valuable, albeit unintended, benefit which also had knock-on effects for year 2, as partners came up with new challenges to propose. In year 2, we saw similar cross-pollination, and a desire for partners to group together to develop some of the solutions further. Nevertheless, incorporating mechanisms to further enhance direct impacts for partners and to research potential gains merits exploration moving forward. We believe that this will require more research by our institution across the years beyond what the students can deliver for the benefits to the public sector and AWS to be made clearer.

Overall, the programme provides a strong foundation for mutually beneficial collaboration between academia, industry, and the public sector. Continued refinement informed by ongoing evaluations will further strengthen the programme's impact.

## VI. CONCLUSION

In conclusion, developing an impact-focused group project module required thoughtful design and continuous improvement through multiple iterations. Key factors for success included establishing strong partnerships in a tripartite fashion between our academic institution, a technology company, and public sector partners. Other factors included embedding responsible innovation into the curriculum, providing authentic professional experiences for students, and actively soliciting feedback from all stakeholders regularly throughout the process. While challenges emerged around

managing expectations, providing adequate support, and handling sensitive data, incremental enhancements based on evaluations aim to address these limitations. This collaborative module provides a promising approach to delivering technical solutions for the public sector while simultaneously enriching student learning and cultivating essential professional skills. Further research into optimising such partnerships and their ultimate impact would be beneficial. Overall, with careful coordination among academic institutions, industry partners, and public sector organisations, programmes like this can enable impact-driven and responsible innovation.

## REFERENCES

- [1] S. Fincher and M. Petre, "Project-based learning practices in computer science education," in *FIE'98. 28th Annual Frontiers in Education Conference. Moving from 'Teacher-Centered' to 'Learner-Centered' Education. Conference Proceedings*, 1998, vol. 3: IEEE, pp. 1185-1191.
- [2] J. Straub, S. Kerlin, and D. Whalen, "Teaching software project management using project based learning (PBL) and group projects," in *2017 IEEE International Conference on Electro Information Technology (EIT)*, 2017: IEEE, pp. 016-021.
- [3] R. Pucher and M. Lehner, "Project based learning in computer science—a review of more than 500 projects," *Procedia-Social and Behavioral Sciences*, vol. 29, pp. 1561-1566, 2011.
- [4] R. Mitchell and P. Parslow, "Group work in computer science." <https://sites.reading.ac.uk/t-and-l-exchange/2018/07/06/group-work-in-computer-science/> (accessed 7/04/2024).
- [5] Challenge Based Learning. "Challenge Based Learning." <https://www.challengebasedlearning.org/> (accessed 07/04/2024).
- [6] L. Johnson and S. Brown, "Challenge based learning: The report from the implementation project," The New Media Consortium, 2011.
- [7] A. R. Santos, A. Sales, P. Fernandes, and M. Nichols, "Combining challenge-based learning and scrum framework for mobile application development," in *Proceedings of the 2015 ACM conference on innovation and technology in computer science education*, 2015, pp. 189-194.
- [8] M. Spezialetti, "Putting people in the picture: Building requirements gathering, design specification and communication skills with video scenarios," in *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*, 2021, pp. 129-135.
- [9] G. I. Allen, "Experiential learning in data science: Developing an interdisciplinary, client-sponsored capstone program," in *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*, 2021, pp. 516-522.
- [10] E. W. McGrath, S. Lowes, C. Jurado, and A. F. Squires, "SE Capstone: A Pilot Study of 14 Universities to Explore Systems Engineering Learning and Career Interest through Department of Defense Problems," in *2011 ASEE Annual Conference & Exposition*, 2011, pp. 22.1277. 1-22.1277. 30.
- [11] L. Adams, M. Daniels, A. Goold, O. Hazzan, K. Lynch, and I. Newman, "Challenges in teaching capstone courses," *ACM SIGCSE Bulletin*, vol. 35, no. 3, pp. 219-220, 2003.
- [12] M. L. Fioravanti *et al.*, "Integrating project based learning and project management for software engineering teaching: An experience report," in *Proceedings of the 49th ACM technical symposium on computer science education*, 2018, pp. 806-811.
- [13] P. Ciancarini, R. Giancarlo, and G. Grimaudo, "Digital Transformation in the Public Administrations: a Guided Tour For Computer Scientists," *IEEE Access*, vol. 12, pp. 22841-22865, 2024.
- [14] G. Dodig-Crnkovic, "Preparing next generation of software engineers for future societal challenges and opportunities," in *Proceedings of the 7th International Workshop on Social Software Engineering*, 2015, pp. 49-52.
- [15] R. Heldal *et al.*, "Sustainability competencies and skills in software engineering: An industry perspective," *Journal of Systems and Software*, p. 111978, 2024.
- [16] A. Orchard and M. O'Gorman, "Fostering responsible innovation with critical design methods," *Journal of Responsible Innovation*, vol. 11, no. 1, p. 2318823, 2024.
- [17] J. Stilgoe, R. Owen, and P. Macnaghten, "Developing a framework for responsible innovation," in *The Ethics of Nanotechnology, Geoengineering, and Clean Energy*: Routledge, 2020, pp. 347-359.



# Serious Game for Industrial Cybersecurity: Experiential Learning through Code Review

Andrei-Cristian Iosif\*, Ulrike Lechner†, Maria Pinto-Albuquerque‡, Tiago Espinha Gasiba\*

\*Siemens AG, Munich, Germany — {andrei-cristian.iosif, tiago.gasiba}@siemens.com

†Universität der Bundeswehr München, Munich, Germany — ulrike.lechner@unibw.de

‡Instituto Universitário de Lisboa (ISCTE-IUL), ISTAR, Lisbon, Portugal — maria.albuquerque@iscte-iul.pt

**Abstract**—Every stage of the industrial software development process is crucial for ensuring high-quality results in a time of increasing digitalization and complexity. Code review is a method to enhance software quality and also promote knowledge exchange among teams. It is generally accepted that the earlier that software bugs and vulnerabilities are caught during product development, the more costs can be saved. As such, code review can play an important role in industrial software development.

However, industry experience showcases that code review can be resource-intensive, and the direct impact on code quality can be hard to quantify. Related work shows that practitioners performing code reviews do not focus specifically on security, partly due to a gap in awareness of the topic.

Our research focuses on improving the efficiency and effectiveness of code review practices, particularly in identifying and addressing security issues in an industrial context.

The present work showcases results from using a serious game as a means to empower developers, by exhibiting code review best practices and raising awareness of security concerns. We collect results over a series of 11 experiments conducted in an industrial setting together with a total of 175 industrial practitioners, serving as a pilot stage, based on which we discuss and conclude on important aspects of the design of the game.

**Index Terms**—Code Review, Cybersecurity, Compliance, Action Design Research, Developer Empowerment

## I. INTRODUCTION

Cybersecurity attacks continue to rise in number and impact, with cybercrime costs expected to exceed \$10B per year, according to a study by Forbes in 2023 [1]. To counter this trend, companies secure the software development lifecycle (SDLC) and employ the DevSecOps paradigm, which lays out practices to integrate security into all steps of the SDLC [2].

One of the cornerstones of DevSecOps is shifting security *left* to avoid vulnerabilities in software at earlier stages of development and reduce the costs of remediating the flaw. While DevSecOps focuses on automating security activities, the human factor cannot be discarded. One of the ways to secure development at earlier stages includes manual effort. Securing the development stage through multiple strategies ensures that more bugs are spotted earlier in the lifecycle, as per Rajapakse et al. [3].

One way to improve the level of security of code in the earlier stages of software development is by performing manual code reviews. Code review is the practice of manual source code inspection by developers other than the author [4].

However, according to Braz et al. [5], the lack of security awareness from the developers' side and the lack of resources

are the main causes impacting the effectiveness of code review toward detecting vulnerabilities and generally increasing code quality. Although the security awareness challenge is known, limited studies have been conducted on code review practices with the objective of enhancing cybersecurity. We thus motivate our research to explore increasing the quality of code review practices towards cybersecurity.

**Problem Statement.** This work proposes an innovative approach to code review as part of the SDLC, aimed at raising the level of industrial standards compliance, strengthening secure coding awareness, enhancing the quality of code review, and empowering software developers for effective code reviews.

To this end, we propose a serious game (i.e., a game with an end goal other than pure entertainment [6]) for leveraging code review practices in the industry and enhancing developers' awareness of security best practices.

The design process of the serious game artifact is done in an iterative process, following Action Design Research principles. Drawing on insights from professional experience, this study initiates the development of a serious game prototype aimed at both evaluating and improving secure code review practices.

The paper is organized as follows: Section II introduces related work. Section III follows by describing the design methodology and its reflection within the current implementation. Section IV presents the experimental setup and evaluation strategies. Results are discussed in section V. Finally, Section VI concludes the paper and lays out plans for further work.

## II. BACKGROUND AND RELATED WORK

This section presents an array of findings from selected academic works, and gray literature meant to outline the current challenges of the SDLC in terms of ensuring security through efficient code review.

Our problem statement is positioned at the intersection of four core areas: security, industry standards, awareness, and empowerment. Whereas the first two areas are mainly technical (although technically speaking they should also consider human factors), the latter two consider the human aspect of our research. We borrow from Hänsch et al. [7] and Tessem et al. [8] for the definitions of, respectively, awareness, and empowerment.

### A. Awareness and Empowerment

Hänsch et al. [7] define IT security awareness across three dimensions: *perception* (knowledge of existing software vul-

nerabilities), *protection* (knowing how to secure software), and *behaviour* (intent to perform security activities).

Empowerment in the organizational context is delineated through structural and psychological dimensions, as per Tessem et al. [8]. It comprises power, information, knowledge, reward, and cognitions of meaningfulness, competence, self-determination, and impact. This multifaceted approach underlines the interaction between an individual's access to resources and their intrinsic motivational states. Empowerment can thus be loosely defined as fostering an environment conducive to enhanced autonomy and effectiveness within the workplace.

We utilize both perspectives, technical and human, to address the real-world problem of cybersecurity awareness and strive for the adoption of our ideas by users and in industrial software engineering contexts. Our artifact aims to provide practitioners with insight into the first two dimensions of awareness, aiming to empower them and positively shape their future behavior.

We chose serious games as a medium for information delivery. Previous work has indicated they are effective in disseminating takeaways to users of such an artifact, as laid out in an experience report of Namin et al [9] and through a literature review conducted by Hendrix et al [10].

To better leverage the potential of serious games as a vector of developer empowerment, we must first understand the difficulties and requirements that industrial code review falls under.

### B. Challenges associated with Code Review

Automation provided through DevSecOps can help to lessen human tasks in software testing, as noted by Sánchez-Gordón et al. [2]. Nevertheless, tools like Static Code Analysis (SAST) often produce false positives and thus require human filtration, as per Mao et al. [11].

Ami et al. [12] investigate practitioners' assumptions and behaviors of SAST tools and highlight an industrial tendency of tool over-reliance and lack of awareness on *false negatives*. Rindell et al. [13] explore how enriching the results from SAST tools alone can be insufficient and propose manual code review as one of the ways to tackle security-related tech debt. Tackling false negatives from SAST tools can be tedious, as the tools are unfit to catch, e.g., business logic or architectural security flaws within source code.

Kupsch et al. [14] emphasize that, despite the benefits of automation, manual reviews are still required to ensure good coverage to find and address vulnerabilities.

Bosu et al. [15] suggest that industry professionals find reviews valuable mainly when they enhance code quality. MacLeod et al. [16] explored what makes a code review useful to code contributors, emphasizing challenges like misguided review focus, process ambiguity, and insufficient training.

Recent developments in artificial intelligence (AI) through breakthroughs in Large-Language Models (LLMs) and their application show promise in offering more robust decision-supporting tooling, as highlighted by Fu et al. [17]. Nonethe-

less, given the nascent stage of the LLM field, it would be premature to *exclusively* commit to a tool-centric approach without also empowering with other means the developers who employ an AI-assisted workflow.

Challenges of code review highlighted by MacLeod et al. [16] pertain wholly to the human aspect of code review (e.g. scope ambiguity, timeliness) which further underscores the industrial significance of our work.

### C. Compliance in the Industry

Ramaj et al. [18] highlight that compliance is currently a blind spot within the DevSecOps paradigm. Regarding standardization within the industry, Moyon et al. [19] have highlighted that integrating security into agile software development presents practical challenges. This is especially the case in heavily regulated industries, where scaling agile practices and achieving compliance with security standards leads to enhanced complexity.

Relevant to our work, the IEC 62443-4-1 [20] and IEC 62443-4-2 [21] standards highlight the importance of secure code review in the development lifecycle and the need for specialized knowledge on secure code review among reviewers. ISO/IEC 20246 [22] offers a generic framework for work product reviews applicable across various organizational roles.

Additionally, the ISO/IEC TR 24772-1 [23] provides guidance on avoiding programming vulnerabilities, while standards like MITRE's Common Weakness Enumeration (CWE) Top 25 [24] and OWASP Top 10 [25] offer insights into secure coding practices.

The role played by standards in our work is two-fold. Firstly, they provide a foundation for emphasizing the significance of cybersecurity in software, as standard-compliant products experience enhanced value. Secondly, standards serve as a fundamental resource from which requirements can be extracted and transformed into practical guidelines for reviewers, thereby offering guidance in the pursuit of secure code review within an industrial environment.

## III. METHODOLOGY

The present work offers a preliminary set of findings collected from crafting and assessing a serious game software artifact focused on empowering software developers on secure code review. Starting from a problem formulation, i.e. *"How to empower developers towards performing secure code review?"*, our goal is to iteratively gain knowledge into this industrial challenge while refining our proposed solution – a software artifact in the form of a serious game.

### A. Artifact Design

Our research approach is anchored in the Action Design Research (ADR) method by Sein et al. [26], which focuses on creating and assessing IT artifacts within an organizational context to address specific problems. The authors contend that design science, through the creation of innovative artifacts, aims to enhance human capabilities.

ADR emphasizes iterative collaboration between researchers and practitioners, intertwining problem understanding, action, and evaluation. It operationalizes the principles of Design Science Research (DSR) by embedding the artifact development process within organizational contexts, thus bridging the gap between theory and practice by iteratively designing and evaluating artifacts *in situ*.

Vom Brocke et al. build upon DSR notions laid out by Hevner et al. and formalize the six dimensions of DSR [27]. Next, we will delineate these dimensions as they apply to the development of our serious game artifact.

**Problem Description:** Identifying the gap in current code review training methods – Developers are not effectively equipped with the necessary skills and best practices for security-oriented code review.

**Input Knowledge:** Existing literature on code review processes, design theories, and development practices.

**Research Process:** Designing and developing the software artifact.

**Key Concepts:** Code quality standards, review techniques.

**Solution Description:** Features and functionalities of the software artifact – interactive code review exercises, feedback mechanisms, and tools integration.

**Output Knowledge:** Evaluating the effectiveness of the serious game in improving code review skills among developers.

Situated in an industrial software setting where cybersecurity is vital, our goal is to enhance code review effectiveness and heighten security awareness. Key to our approach is the design and assessment of a serious game for software developers. With two researchers possessing extensive experience in industrial software development and secure coding, we collaborate with the industry to introduce fresh academic perspectives and rigorously evaluate existing practices. Previous work performed by the authors has already shown that embedding a serious game within workshops performed in the industry was well-received [28]. While gamification has been applied to static code analysis and secure coding in previous works, our approach uniquely integrates these elements into a comprehensive serious game designed to enhance secure review practices in an industrial context.

Based on the positive findings of the first ADR iteration, we consider further refining the artifact to better understand the problem of secure code review and address the practitioners' needs on the topic. We are also looking into the *resource* problem - code review is a demanding process on teams, and therefore we need an approach to optimize the necessary resources. Collaborating with industry practitioners is, therefore, essential for collecting insight into this matter.

Our methodology adopts an iterative stance specific to ADR, with the iteration continuing until a "satisficing" threshold is attained [29]. The iteration process persists until the developed solution adequately addresses the key issues within the problem context, meeting the essential needs and expectations of stakeholders to a degree considered satisfactory and functional – i.e. when participants consistently report that they draw knowledge and practical applicability from participating

in the proposed serious game, essentially heightening their empowerment in performing secure code review.

In this first iteration, we aim to evaluate the design in terms of providing developers with the awareness and necessary resources for performing a proper code review for security features. Our artifact aims at improving secure code review, by creating awareness and individual empowerment.

Up to and including this section of the paper, the present work has encompassed discussions that introduced aspects related to the first four dimensions of DSR. In the following subsection, we bring the serious game artifact into the spotlight and present a high-level implementation overview, thus touching on the fifth dimension, the *Solution Description*.

### B. Implementation

We propose a serious game targeted at developers in the industry to serve as an educational tool geared towards empowering practitioners in performing secure code reviews.

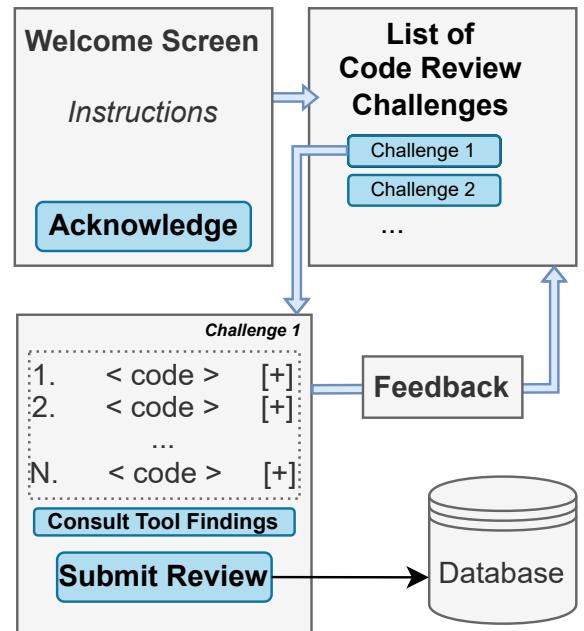


Fig. 1. Artifact Overview

Figure 1 presents our first approach to the game's design. Instructions for the platform are provided to the user on the welcome page. After acknowledging the instructions, they are presented with a list of code review challenges. Upon selecting a challenge, the user receives a source code snippet containing software vulnerabilities and bad practices. The user can select parts of the code and annotate them with comments.

Our serious game encompasses 28 exercises, covering 4 programming languages – Java, C#, Python, and JavaScript. The vulnerabilities included in the game's snippets cover the CWE Top25 [24] and OWASP Top10 [25], to address the most prevalent vulnerabilities encountered in software.

Figure 2 presents the main interface of the serious game. The interface is structured in tabular form, over four columns: **Comment**, **Line Number (#)**, **Code** and **Solution**.



Fig. 2. Artifact Main Interface

The user enters his review comments in the first column. Findings from SAST tools can be consulted in parallel to the source code. By including SAST output in the serious game, we aim to achieve two objectives: firstly, the users are expected to gain familiarity and expertise with using SAST findings in their code review (emulating a real-world industrial tooling environment); secondly, the user is educated on the notions of *false positives/negatives*. By observing the tools' shortcomings or overzealousness, developers can understand where human rationalizing is needed.

When done, the user submits their review, and all their interactions are pushed to a database for later inspection.

Upon finding at least 50% of the vulnerabilities in the snippet, the full intended solution is displayed in the third column for the user to reflect upon, which serves as a direct means of feedback. This percentage stems from a previous work where the authors conducted an experiment consisting of three separate industry interventions [28] in which a simplified version of the proposed artifact was trialed –we observe that developers who undergo security training are able, on average, to identify approximately half of the vulnerabilities that an expert might detect.

The players' progress and ranking is tracked with a dashboard, also accessible to the players, to enhance the game aspect and foster competitiveness.

#### IV. EXPERIMENT

This section describes the experimental setup through which we evaluate the artifact.

##### A. Industrial Intervention

We deployed our serious game artifact throughout a series of industrial interventions. In the context of our study, an *intervention* refers to a workshop on security topics in which the proposed serious game has been embedded for experiential learning.

Across 11 events, 175 participants were presented with the serious game, of which 47 provided feedback about the artifact. Events consisted of secure coding workshops (online, in Germany, and in China), from February 2023 to January 2024. Participants were aged 24-65 and 94.12% graduated from higher education. The vast majority (88.12%) reported an agile working style.

The content was tailored to the participants' areas of expertise. The first two days are spent educating participants on common security antipatterns and their respective mitigations, through a combination of presentations, live demonstrations, and hands-on exercises.

The final day consists of a capture-the-flag-style event where the participants compete against each other in solving various types of challenges – offensive (exploiting binaries or websites), defensive (patching vulnerable programs), and challenges on secure code review, where the participants are presented with the aforementioned artifact and are expected to be able to spot vulnerabilities. The participants could freely choose what types of challenges they wanted to solve.

##### B. Evaluation and Validation

One of our goals is to foster, through our SG, an environment conducive to enhanced autonomy and effectiveness within the workplace, both being features of empowerment. By assessing usability and perceptions of specific review challenges, we aim to understand how these factors contribute to the empowerment of developers in performing secure code reviews.

The developed artifact is evaluated in terms of usefulness and enjoyability (Graziotin et al. observed that "*happy programmers are good programmers*" [30]).

Our study uses a semi-structured interview method, following Wilson et al. [31], which delivers questions aimed at individually assessing the platform's success in empowering the user to perform secure code reviews. Participants are asked to fill out an anonymous survey. The survey is conducted voluntarily, via Microsoft Forms, to share how they perceive the serious game.

The collected data is devoid of any personally identifiable information, and as such, our analysis focuses on aggregated results from a diverse pool of participants rather than individual-level data. This approach aligns with ethical research practices and industry standards.

#### V. RESULTS AND DISCUSSION

Figure 3 illustrates levels of agreement on game aspects, following a Likert-like scale [32], obtained from the survey responses. **Q1** and **Q3** give insight into how respondents assess their skill in identifying vulnerabilities, within the game's context and professionally. **Q2**, **Q4** and **Q5** explore the participants' reception of the game itself on three implementation aspects - length adequacy, usability, and clarity of instructions.

A substantial majority of respondents concurred that the length of the code snippets was adequate (86.4%, Q2), with a similarly high percentage agreeing that the game's design was usable (81.2%, Q4) and that its instructions were clear (83.3%, Q5).

Conversely, a significant portion of the participants expressed difficulty in identifying vulnerabilities within the code snippets (57.7% combined for "Disagree" and "Neutral," Q3).

Addressing the familiarity with the vulnerabilities presented, the majority stance was neutral, with 60% of participants

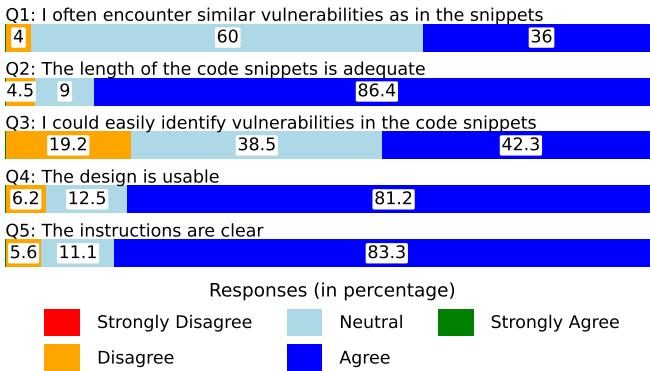


Fig. 3. Agreement Levels towards the Serious Game

neither agreeing nor disagreeing that they often encounter similar vulnerabilities in their work (Q1). Similarly, more than a third of the respondents are neutral in saying that they could easily identify the vulnerabilities presented in the game (Q3, 38.5%). Furthermore, nearly 1 in 5 of the respondents express difficulty in identifying vulnerabilities within the game (Q3, 19.2%). Together, these percentages may hint towards a lack of security awareness and experience of the participants, which may be the scope of further investigative work.

Our design shows that the chosen length of the snippets is positively regarded (86.4%, Q2). The median length of the exercises in the platform is 44 lines of code, with the longest snippet spanning across 116 lines and the shortest across 12. More than 4 out of 5 survey participants report agreement toward designing exercises of this length, this being the highest percentage of agreement recorded in our survey.

It is interesting to note that 19.2% of respondents said that disagreed about being able to easily identify vulnerabilities within the serious game (Q3), while only 4% expressed disagreement towards encountering similar vulnerabilities in their profession (Q1). Based on this difference, we have indicators to believe that this may be due to a lack of awareness of cybersecurity and how to perform reviews for it. Nonetheless, combining the disagreement and neutrality percentages, we reach more similar figures for the two statements – 64% for Q1 and 57.7% for Q3. This is in line with expectations, as the presented vulnerabilities were purposefully selected to cover the most prevalent security defects, as covered in Section III. Nonetheless, further work would be required to explore how the participants' perception differs between the production code and the serious game's snippets.

Concerning feedback on the platform itself, including challenges, we have collected positive feedback (Q2, Q4, Q5). The descriptions of the tasks for each challenge are perceived as being clear, the design is deemed to be usable, and the length of the code snippets is considered to be adequate. These collected responses serve to enhance the trust in the platform and prompt us to contemplate expanding it with additional challenges and further refining the feedback mechanism.

Based on data collected from the industrial interventions,

we extend findings from our previous study [28] across a wider range of programming languages and tools. Our present work demonstrates the broader applicability of the following statement: participants receiving minimal security training consistently outperform SAST tools. This is measured through the solve rate of the challenges. By design, the exercises showcase the dangers of tool over-reliance via *false-negatives*. Observing successful solutions, i.e. users finding 50% or more defects, we can imply that our serious game is useful for helping developers be self-reliant and perform effective code reviews, as solving the challenges requires being able to outperform the tools.

Our collected findings from the first set of experiments, together with the feedback from informal discussions with participants, provide a good indicator of the suitability of our design as a means to address secure code review in an industrial context. This aligns with our experience in the field.

Research in the industry comes with limiting factors. Designing a game can be characterized as a *wicked problem*, as per Baskerville [33] and Hevner [29]. The implication is that there is not one optimal solution, and the usefulness of the resulting artifact is influenced by the surrounding social environment as well as social and cognitive capacities. Ensuring a *control group* to evaluate the artifact is also unfeasible in industrial settings, due to these same factors.

The drawn conclusions may be influenced by a restricted number of participants, their backgrounds, functions, and experience levels. Predominantly positive findings may be explained by the survey's voluntary nature and the participant count; those with unfavorable views might have opted out, and awareness of the study's intent might incur positive bias.

However, our findings align with prior research with comparable feedback from both industrial and academic sectors. Thus, we believe that our past and future conclusions would remain largely unchanged even with a larger participant pool. While preliminary indicators point towards the game's suitability, our results are still limited, thus requiring further steps in the design and evaluation process.

## VI. CONCLUSION AND FURTHER WORK

With the industry transitioning towards integrating security into every step of the software lifecycle, challenges arise in efficiently including adequate practices to assure software quality. One way to tackle code quality is to raise awareness and empower developers on the topic of secure code review. In the present work, we propose to achieve this through a serious game.

This work presents a serious game to raise awareness of code review and its design. We present the motivation behind the work, the importance of the topic in the industrial context in which the work is being carried out, as well as preliminary results from a first design cycle iteration.

We detail how we structure our deployment of the game and show that it can be effectively implemented in diverse settings, with participant pools of varying backgrounds. This facilitates the replication of our study through the repeatability

of the experiment and highlights the serious game's broader application for education and learning.

Our research addresses a pressing industrial requirement: empowering developers with the necessary tools and knowledge for secure code review.

The proposed design of the artifact shows promise. Conclusions drawn from the survey show strong indicators that its format, instructions, and snippets are positively perceived.

In the future design cycles of the work, authors would like to further refine the feedback mechanism, assisting information delivery by employing visual cues and enhancements to the format and broadening the application scope by integrating more programming languages and tools into the artifact.

Good code reviews can only be done with an awareness of secure coding practices and with the necessary structures and the determination to achieve quality code reviews. Participants reported, through empirical evaluation, that the experiential learning activity, conveyed through the serious game, improved their awareness of secure code review. The gained knowledge, specifically towards better recognizing hidden faults in source code, is perceived as useful.

Based on the artifact's positive reception within the industrial target audience, we have reasons to believe that the proposed serious game can be used to foster awareness of secure code review and provide the impulse to create conditions conducive to organizational support for meaningful code review practices.

## REFERENCES

- [1] C. Brooks, "Cybersecurity trends & statistics for 2023; what you need to know," [bit.ly/forbes23cyber](https://bit.ly/forbes23cyber), 2023, "Online, Accessed 2023.10.02".
- [2] M. Sánchez-Gordón and R. Colomo-Palacios, "Security as culture: A systematic literature review of DevSecOps," in *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops*, ser. ICSEW'20. New York, NY, USA: Association for Computing Machinery, 06 2020, pp. 266–269.
- [3] R. N. Rajapakse, M. Zahedi, M. A. Babar, and H. Shen, "Challenges and solutions when adopting devsecops: A systematic review," *Information and Software Technology*, vol. 141, p. 106700, 2022.
- [4] J. Cohen, *Modern code review*. Sebastopol, CA, USA: O'Reilly, 2010.
- [5] L. Braz and A. Bacchelli, "Software security during modern code review: the developer's perspective," in *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. Singapore: ACM, Nov. 2022, p. 810–821.
- [6] T. Susi, M. Johannesson, and P. Backlund, "Serious games: An overview," IKI Technical Reports, Tech. Rep., 2007.
- [7] N. Hänsch and Z. Benenson, "Specifying IT Security Awareness," in *Proceedings - International Workshop on Database and Expert Systems Applications, DEXA*. Munich, Germany: IEEE, 12 2014, pp. 326–330.
- [8] B. Tessem, "Individual empowerment of agile and non-agile software developers in small teams," *Information and software technology*, vol. 56, no. 8, pp. 873–889, 2014.
- [9] A. Siami Namin, Z. Aguirre-Muñoz, and K. Jones, "Teaching cyber security through competition an experience report about a participatory training workshop," in *7th Annual International Conference on Computer Science Education: Innovation & Technology (CSEIT 2016)*, ser. CSEIT. Global Science & Technology Forum (GSTF), Oct. 2016.
- [10] M. Hendrix, A. Al-Sherbaz, and V. Bloom, "Game based cyber security training: are serious games suitable for cyber security training?" *International Journal of Serious Games*, vol. 3, no. 1, Mar. 2016.
- [11] R. Mao, H. Zhang, Q. Dai, H. Huang, G. Rong, H. Shen, L. Chen, and K. Lu, "Preliminary Findings about DevSecOps from Grey Literature," in *2020 IEEE 20th International Conference on Software Quality, Reliability and Security*. Macau, China: IEEE, 2020, pp. 450–457.
- [12] A. S. Ami, K. Moran, D. Poshyvanyk, and A. Nadkarni, "False negative – that one is going to kill you: Understanding industry perspectives of static analysis based security testing," *preprint arXiv:2307.16325*, 2023.
- [13] K. Rindell, K. Bernsmid, and M. G. Jaatun, "Managing security in software: Or: How I learned to stop worrying and manage the security technical debt," in *Proceedings of the 14th International Conference on Availability, Reliability and Security*. New York, NY, USA: Association for Computing Machinery, 2019, pp. 1–8.
- [14] J. A. Kupsch and B. P. Miller, "Manual vs. Automated Vulnerability Assessment: A Case Study," in *First International Workshop on Managing Insider Security Threats (MIST)*. New York, NY, USA: Association for Computing Machinery, 06 2009, pp. 83–97. [Online]. Available: [pages.cs.wisc.edu/~kupsch/va/ManVsAutoVulnAssessment.pdf](http://pages.cs.wisc.edu/~kupsch/va/ManVsAutoVulnAssessment.pdf)
- [15] A. Bosu, M. Greiler, and C. Bird, "Characteristics of Useful Code Reviews: An Empirical Study at Microsoft," in *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*. Florence, Italy: IEEE, 2015, pp. 146–156.
- [16] L. MacLeod, M. Greiler, M.-A. Storey, C. Bird, and J. Czerwonka, "Code reviewing in the trenches: Challenges and best practices," *IEEE Software*, vol. 35, no. 4, pp. 34–42, 2017.
- [17] M. Fu and C. Tantithamthavorn, "Linevul: a transformer-based line-level vulnerability prediction," in *Proceedings of the 19th International Conference on Mining Software Repositories*. Pittsburgh PA USA: ACM, May 2022, p. 608–620.
- [18] X. Ramaj, M. Sánchez-Gordón, V. Gkioulos, S. Chockalingam, and R. Colomo-Palacios, "Holding on to compliance while adopting devsecops: An slr," *Electronics*, vol. 11, no. 22, pp. 1–19, 2022.
- [19] F. Moyon, D. Mendez, K. Beckers, and S. Klepper, "How to integrate security compliance requirements with agile software engineering at scale?" in *Product-Focused Software Process Improvement*, M. Morisio, M. Torchiano, and A. Jedlitschka, Eds. Cham: Springer International Publishing, 2020, pp. 69–87.
- [20] ISO/IEC 64223-4-1:2018-1, "ISO/IEC 62443-4-1:2018 Security for industrial automation and control systems - Part 4-1: Secure product development lifecycle requirements," International Organization for Standardization, Geneva, CH, Standard, 1 2018.
- [21] ISO/IEC 64223-4-2:2019-12, "Security for Industrial Automation and Control Systems - Part 4-2: Technical Security Requirements for IACS Components," International Electrical Commission, Geneva, CH, Standard, 1 2019, ISBN 978-2-8322-6597-0.
- [22] ISO/IEC 20246:2017, "Software and systems engineering – Work product reviews," International Organization for Standardization, Geneva, CH, Standard, 2017.
- [23] ISO/IEC TR 24772-1:2019, "Programming languages – Guidance to avoiding vulnerabilities in programming languages – Part 1: Language-independent guidance," International Organization for Standardization, Geneva, CH, Standard, 2019.
- [24] "CWE Top 25 Most Dangerous Software Weaknesses," <http://bit.ly/mitre25>, MITRE, 2023, online, accessed 2023.07.24.
- [25] O. Foundation, "OWASP Top 10 - 2021," <https://owasp.org/Top10>, OWASP Foundation, 2021, online, accessed 2023.07.24.
- [26] M. K. Sein, O. Henfridsson, S. Purao, M. Rossi, and R. Lindgren, "Action Design Research," *MIS Quarterly*, vol. 35, pp. 37–56, 2011.
- [27] J. Vom Brocke and A. Maedche, "The DSR grid: six core dimensions for effectively planning and communicating design science research projects," *Electronic Markets*, vol. 29, no. 3, p. 379–385, Sep. 2019.
- [28] A.-C. Iosif, T. Espinhal Gasiba, U. Lechner, and M.-P. Albuquerque, "Raising awareness in the industry on secure code review practices," in *CYBER 2023: The Eighth International Conference on Cyber-Technologies and Cyber-Systems*. IARIA, September 2023, pp. 62–68.
- [29] A. Hevner, S. March, J. Park, and S. Ram, "Design Science in Information Systems Research," *Management Information Systems Quarterly*, vol. 28, pp. 75–, 03 2004.
- [30] D. Graziotin, F. Fagerholm, X. Wang, and P. Abrahamsson, "What happens when software developers are (un)happy," *Journal of Systems and Software*, vol. 140, pp. 32–47, 2018.
- [31] C. Wilson, "Semi-Structured Interviews," in *Interview Techniques for UX Practitioners*. Burlington, MA, USA: Elsevier, 2014, pp. 23–41.
- [32] R. Likert, "A Technique for the Measurement of Attitudes." *Archives of psychology*, vol. 22, no. 140, pp. 1–55, 6 1932. [Online]. Available: [bit.ly/lkert](http://bit.ly/lkert)
- [33] R. L. Baskerville and A. T. Wood-Harper, "A critical perspective on action research as a method for information systems research," *Journal of Information Technology*, vol. 11, no. 3, pp. 235–246, Sep. 1996.

# **Journal First Track**



# Towards understanding students' sensemaking of test case design: a one-page summary

Niels Doorn  \*<sup>†</sup>, Tanja E. J. Vos  \*<sup>‡</sup>, Beatriz Marín  <sup>‡</sup>

\*Open Universiteit *{niels.doorn,tanja.vos}@ou.nl*

<sup>†</sup>NHL Stenden University of Applied Sciences *niels.doorn@nhlstenden.com*

<sup>‡</sup>Universitat Politècnica de València *{tvos,bmarin}@dsic.upv.es*

**Abstract**—This study examines sensemaking in student test case design, showing a reliance on conceptual knowledge learned during programming courses over exploratory testing strategies. Of the three identified approaches taken by students, the “developer approach” is used most often, suggesting a gap in software engineering education. We hypothesise that software testing should be taught in computer science programs using a design paradigm based on empiricism instead of rationalism. Based on these results, and our hypothesis, we will further analyse the sensemaking processes of both students and experts, and create an instructional design to improve software testing education in computer science programs.

## I. INTRODUCTION

Within a society where the role of software systems increases, the need for the quality of these systems becomes correspondingly important. In software engineering practices in the industry, software testing is generally accepted as one of the most effective methods to measure the quality of a software system. Despite this, software engineering education fails to train graduates with the skills, conceptual knowledge and attitude in such a way that they meet the industry expectations and needs [1]–[3].

In order to create a instructional design on software testing applicable in different computer science educational contexts, we first need to gain a holistic view [4] of students cognitive processes when testing software, and the different approaches taken.

## II. OBJECTIVE

Before we can design interventions to improve our pedagogical approaches, we need to gain more in-depth understanding and recognition of sensemaking as it is happening when students design test cases.

## III. METHODOLOGY

After an initial exploratory study [5] in which we identified four different approaches taken by students, we did a follow-up study with 50 students. These students were given a description of a system under test and where asked to design tests for this system using a web-based tool for modeling test cases. We collected the models, their metadata, recordings of the computer screen while the models were created, and we used a questionnaire to study their perspectives on the assignment. We analysed how well the created models fit the system under test, the sensemaking process that students

went through when creating the models, and the students' perception of the modeling task using an iterative inductive analysis process to allow new information about the different sensemaking approaches to emerge.

## IV. RESULTS

The findings in this study show the existence of different sensemaking approaches taken by student. The most common approach is the ‘developer approach’ where the students use constructs similar to those of programming to model the problem. Only in the least used divergent tester approach we see students who create new knowledge about the domain of the problem under test, and also on the design of test cases.

## V. CONCLUSIONS

Based on these results, we can further investigate ways to influence the sensemaking process in education, leading to an instructional design for software testing in software engineering education. Our first steps are to gain more insights in the sensemaking process taken by test experts.

## VI. ORIGINAL ARTICLE

This one-page summary is based on an original article [6] published in Data & Knowledge Engineering, Volume 146, July 2023

## REFERENCES

- [1] L. Scatalon, J. Carver, R. Garcia, and E. Barbosa, “Software testing in introductory programming courses,” in *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, ser. SIGCSE ’19. New York, NY, USA: ACM, Feb. 2019, pp. 421–427. [Online]. Available: <https://doi.org/10.1145/3287324.3287384>
- [2] V. Garousi, A. Rainer, P. Lauvås Jr, and A. Arcuri, “Software-testing education: A systematic literature mapping,” *Journal of Systems and Software*, vol. 165, p. 110570, 2020.
- [3] Y. Lizama, D. Varona, P. Waychal, and L. F. Capretz, “The unpopularity of the software tester role among software practitioners: a case study,” in *Advances in RAMS Engineering*. Springer, 2020, pp. 185–197.
- [4] T. O. B. Odden and R. S. Russ, “Sensemaking epistemic game: A model of student sensemaking processes in introductory physics,” *Phys. Rev. Phys. Educ. Res.*, vol. 14, p. 020122, Nov 2018. [Online]. Available: <https://link.aps.org/doi/10.1103/PhysRevPhysEducRes.14.020122>
- [5] N. Doorn, T. E. J. Vos, and B. Marín, “Exploring students’ sensemaking of test case design, an initial study,” in *2021 IEEE 21th International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, 2021, pp. 1–6.
- [6] ———, “Towards understanding students’ sensemaking of test case design,” *Data & Knowledge Engineering*, vol. 146, p. 102199, 2023. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0169023X23000599>



# Summary of: Lessons Learned from a Learning Program for Software Architects

Frances Paulisch

*Siemens Healthineers AG*

*Head of Software Initiative*

Forchheim, Germany

frances.paulisch@siemens-healthineers.com

Matthias Backert

*Siemens Healthineers AG*

*Software Initiative, Training*

Erlangen, Germany

matthias.backert@siemens-healthineers.com

Thomas Blum

*Siemens Healthineers AG*

*Diagnostic Imaging, Magnetic Resonance*

Erlangen, Germany

thomas.blum@siemens-healthineers.com

This paper was published in IEEE Software, vol. 40, no. 6, pp. 55-62, Nov.-Dec. 2023, DOI Bookmark: 10.1109/MS.2023.3311267 Full article at:  
<http://ieeexplore.ieee.org/iel7/52/10339104/10242135.pdf>

Today's world is changing quickly in multiple dimensions: technology is becoming a key part of our world, systems are becoming increasingly complex, and the environment is becoming more volatile and uncertain. At the same time, the quality attributes are growing, including attributes of the system like safety, security, availability, and performance but also developmental attributes such as maintainability and testability. Software professionals working on creating such systems not only face these challenges but also must be able to grasp and apply new concepts quickly. This also includes new technological approaches, processes, and business models such as continuous delivery and DevOps, ecosystems, microservices, machine learning, etc., but also how to apply these to the business situation and domain and in the context of long-living systems.

Especially with so many things changing, we find it important to have a strong focus on the architecture of the system and, consequently, the education and training of the architects. Because architecture is such a cross-cutting concern, this allows us to have the most business impact, avoiding the biggest potential problems and enabling, where feasible, a fast pace of change. The architecture focus is also necessary because design and architecture are known as "wicked problems," which implies that there is no single right solution, but one must consider different stakeholder perspectives and evaluate tradeoffs. Therefore, establishing a way of thinking about problems and learning to consider the alternatives and tradeoffs is an extremely useful approach, especially in the area of architecture.

In 2006 a cross-company core team at Siemens (including the healthcare part of the company) started working on this topic and soon realized that for the mentioned reasons a standard off-the-shelf training would not be sufficient for our purposes. Similar to how we develop complex products, we considered our business case, our stakeholders, and the nonfunctional requirements of our "learning program" product, and established a team with good communication and social skills to pilot and grow the first learning program. Through a series of interviews and other means, we gathered information (and still do) on the root causes underlying insufficient attention to software architecture, the characteristics of software architects who have been successful on multiple complex projects in the past, and the key success factors and pitfalls for software architects in the context of our high-quality, often safety-critical, complex cyberphysical systems.

After Siemens Healthineers became a separate company, most aspects of the software architecture learning program were retained and further enhanced. Siemens Healthineers, who had been one of the drivers from the beginning, established some additional aspects and integrated the learning program with our personnel processes and career paths.

The aspects that differentiate our program from other programs described in the literature are that our program

- is focused on professionals in a real-world industrial setting applying their learnings in their current projects
- is aligned to the needs in our context characterized by high-quality cyberphysical systems with a broad range of quality attributes (performance, scalability, security, safety, reliability...) and very long life spans
- is embedded in a company context (e.g., personnel departments, career paths, certification process, and visibility in the company)
- has a strong focus on the active network of the company-internal community (developers, architects, trainers, assessors...)
- is established as the architecture training for the whole company (all business units and global)
- the content systematically evolves over time and the didactics follow an agile learning approach
- has a long (over 15 years) history in industrial practice.

Despite these points that make our program unique, we are confident that many of the insights we share in the paper could apply also for other practitioner-oriented software engineering learning programs.

The software architect learning programs have an over 15-year history, which in the days of software is a very long time. As such, one can learn from how those foundations were built, for example, applying approaches that we use for our medical devices with long life spans and also apply them to our training-related "products" such as this set of learning programs. Similar to how our products are very diverse but also have many elements in common, our set of ca. 250 architects are also strong individuals but, through the learning program, now share a common language and a more systematic way to think about challenges.

The full paper in volume 40 of IEEE Software (<http://ieeexplore.ieee.org/iel7/52/10339104/10242135.pdf>) describes a role-based learning program aimed primarily at software architects and shares lessons that can be applied to the readers' own learning endeavors.



# Extension of a study on Parsons' Problems reveals implications for SEE&T

Kevin D. Wendt

*Department of Computer Science & Eng.*

*University of Minnesota*

Minneapolis, MN, USA

ORCID: 0009-0006-5475-2832

**Index Terms**—Parsons' Problems, Code Mangler, unfamiliar code, code tracing, code writing, new grads

## I. INTRODUCTION

“The Code Mangler: Evaluating Coding Ability Without Writing Any Code” [1] introduced a hypothetical bad actor into an established, but still relatively new, problem type - the Parsons’ Problem [2]. When I replicated and extended the study, similar results to the original were observed: students found that rearranging lines of correct code that were in jumbled order was as difficult and as good at assessing their abilities as writing code from scratch [3]. Not only is this finding helpful in providing an accurate assessment tool with significantly reduced effort required to grade, it presents an interesting possibility for future software engineers: the ability to practice and gain confidence in working with unfamiliar code, a skill which may translate to industry.

## II. RESULTS

Like the original paper, there was a difference between how long it takes to grade Code Mangler problems vs. traditional code-writing problems: 2.8-4.4 times faster grading for Code Mangler problems. Likewise, students who did well on the entire exam did well on the Code Mangler problem, with medium to strong correlation at high statistical significance (Spearman’s Rank Correlation Coefficient ( $r_s = .3108, p < .001$  or  $1e-3$ )). By asking students for their perception of the experience, we gain additional insight to the impact of these questions. The questions, asked once per problem variant, were as follows:

- What level of difficulty was the *<traditional/mangled>* problem?
- How well did the format of the *<traditional/mangled>* problem assess your abilities?
- How confident are you in your answer to the *<traditional/mangled>* problem?

In all three instances - difficulty, ability to assess skills, and confidence in the answer, not only was the “equal” answer the clear winner, but those answers which did favor one variant over the other were balanced. The only statistically significant difference between the variants was with regard to difficulty. Traditional questions were considered to be less difficult than Code Mangler questions ( $t(150) = 2.23, p = .027$ ). While this

meets the lowest bar of statistical significance typically used ( $p < .05$ ), it has been described as being “suggestive” rather than “significant” [4].

## III. DISCUSSION

While the results do provide additional evidence that these problems can be an effective alternative to code-writing exercises, the qualitative results revealed an additional point of interest to software engineering educators: comments which indicate a lack of preparation for industry: “The code mangler seems easier in terms of syntax errors, but understanding the logic of who wrote the program can be harder than writing your own version” and “It felt harder than the writing your own code question...it didn’t test my knowledge of writing code but tested how well I could read others and fix it.”

## IV. CONCLUSION

Originally, Parsons’ Problems were intended to provide “rote learning of syntactic constructs” [2]. This learning through repetition could benefit new grads entering industry. Repetitive practice with Parsons’ Problems could increase the learner’s ability to understand unfamiliar code. An increase may then reduce time-to-productivity in the field when joining a new team as a new developer or as an experienced developer joining a new team. Our future work includes the development of a tool to more easily generate a large number of variations of each Parsons’ Problem which can be imported directly into LMS systems, like Canvas or Moodle. This would allow rote learning, even on code not written by the instructor, with minimal effort required of the instructor.

## REFERENCES

- [1] N. Cheng and B. Harrington, “The Code Mangler: Evaluating Coding Ability Without Writing Any Code”, in *Proc. 48th ACM SIGCSE Tech. Sym. on Comp. Sci. Ed.*, 2017, pp. 123–128, <https://doi.org/10.1145/3017680.3017704>.
- [2] D. Parsons and P. Haden, “Parson’s Programming Puzzles: A Fun and Effective Learning Tool for First Programming Courses”, In *Proc. 8th Australasian Conf. on Computing Ed. (ACE ’06)*, vol. 52, 2006, pp. 157–163, doi:10.5555/1151869.1151890.
- [3] K. D. Wendt, “A Replication of and Extension to the Original Code Mangler Study,” in *IEEE Software*, vol. 40, no. 6, pp. 102–108, Nov.-Dec. 2023, doi: 10.1109/MS.2023.3299268.
- [4] A. Cockburn, P. Dragicevic, L. Besançon, and C. Gutwin, “Threats of a replication crisis in empirical computer science,” *Commun. ACM*, vol. 63, no. 8, pp. 70–79, 2020, <https://doi.org/10.1145/3360311>.



# Thriving in the era of hybrid work: Raising cybersecurity awareness using serious games in industry trainings

Tiange Zhao  
and Tiago Gasiba  
Siemens AG  
Otto-Hahn-Ring 6  
81739, München, Germany  
Email: tiange.zhao@siemens.com  
Email: tiago.gasiba@siemens.com

Ulrike Lechner  
University of the Bundeswehr Munich  
Werner-Heisenberg-Weg 39  
85579, Neubiberg, Germany  
Email: ulrike.lechner@unibw.de

Maria Pinto-Albuquerque  
Instituto Universitário de Lisboa  
(ISCTE-IUL)  
ISTAR, Av. das Forças Armadas  
1649–026 Lisboa, Portugal  
Email: maria.albuquerque@iscte-iul.pt

Our work [1] has been accepted in Journal of Systems and Software. It was first received on 30th January 2023 and revised on 31st October 2023, accepted 23rd December 2023. It is available online since 2nd January 2024; Version of Record 24th January 2024.

**Abstract:** Modern software engineering education aims to prepare software engineers for hybrid work environments. The shift to work-from-home (WFH) or work-from-anywhere (WFA) has increased the importance of cybersecurity. An industrial case study revealed that raising awareness is crucial. We developed two serious games, CyberSecurity Challenges (CSC) and Cloud of Assets and Threats (CATS) to enhance cybersecurity training. These games empower practitioners to address hybrid work challenges while ensuring secure software development and cloud security. Empirical evidence supports the effectiveness of serious games in raising awareness among software professionals.

**Introduction:** Nowadays, an increasing number of the training in the industry are conducted online or hybrid. Hybrid work refers to the work mode where a part of or all of the training participants join the training online instead of staying together in a classroom physically. This work provides a unique industry experience report on adapting the traditional cyber-security training with the enrichment of two novel serious games that we designed under the design science research paradigm.

**Methodology:** Our work is motivated by a case study highlighting the need to establish secure coding guidelines in the industry and raise awareness of cloud security issues. Following the guidance of the design science research paradigm, we designed the first serious game CSC and validated our design with the target group in 15 game events. By reusing the infrastructure and inspired by the success story of the CSC, we designed the second serious game, CATS, which focused on cloud security. We evaluate CATS in a hybrid environment in trainers, trainees, and organizational perspectives in 9 game events with 108 participants.

**Result:** Based on the data we collected, the results show that applying CSC and CATS in training is helpful in raising awareness among participants. According to our semi-structured survey, the result also shows the challenges that trainer faces in hybrid work and serious games can be a solution to those challenges.

**Contribution:** The contribution of our work is that we present the valuable experience in combining the advantage of different training modes and mitigating the disadvantage of online training. We observe the impact of serious games through a scientifically sound approach based on the data and feedback we collected systematically from the trainers', trainees', and the organization's perspectives in the industry. We show through empirical evidence that serious games are a successful approach for training conducted in hybrid work mode while providing authentic and immersed experiences that empower and raise cybersecurity awareness of current and future software professionals.

## REFERENCES

- [1] T. Zhao, T. Gasiba, U. Lechner, and M. Pinto-Albuquerque, "Thriving in the era of hybrid work: Raising cybersecurity awareness using serious games in industry trainings," *Journal of Systems and Software*, vol. 210, p. 111946, 2024. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0164121223003412>

## ACKNOWLEDGMENT

This research task was partially supported by Fundação para a Ciência e a Tecnologia, I.P. (FCT) [ISTAR Projects: UIDB/04466/2020 and UIDP/04466/2020]. Ulrike Lechner acknowledges funding by dtec.bw for project LIONS and dtec.bw is funded by the European Union — NextGenerationEU and for project CONTAIN by the Bundesministerium für Bildung und Forschung (FKZ 13N16581). Tiange Zhao and Tiago Gasiba acknowledge the funding provided by the Bundesministerium für Bildung und Forschung (BMBF) for the project CONTAIN with the number 13N16585.



# Teaching Engineering of AI-intensive Systems

1<sup>st</sup> Atif Mashkoor  
*Johannes Kepler University*  
Linz, Austria  
atif.mashkoor@jku.at

2<sup>nd</sup> Wesley K. G. Assunção  
*North Carolina State University*  
Raleigh, USA  
wguezas@ncsu.edu

3<sup>rd</sup> Alexander Egyed  
*Johannes Kepler University*  
Linz, Austria  
alexander.egyed@jku.at

**Abstract**—With AI increasingly affecting software systems, there is a pressing need to prepare the next generation of software engineers to build AI-intensive systems proficiently. This work outlines our instructional approach in the “Engineering of AI-intensive Systems” course for postgraduate computer science students to bridge the knowledge gap between software engineering (SE) and artificial intelligence (AI) disciplines. Our paper elaborates on the course’s framework, pedagogical strategies, and evaluation methods, emphasizing the benefits of this interdisciplinary educational model.

## I. INTRODUCTION

AI-intensive systems have overtaken the world [1]. However, AI-intensive systems and their engineering are poorly understood [2]. This is made worse by AI and SE being seen as independent fields of knowledge, with different degree programs and sparsely overlapping courses.

To address this limitation, we initiated a novel course titled “Engineering of AI-intensive Systems” to promote collaboration between students from AI and SE. The course’s objectives were twofold: first, introducing AI students to the fundamental principles of SE, and second, providing SE students with practical experience in tackling real-world AI challenges. The core concept was establishing mixed groups comprising AI and SE students. These groups collaborate on addressing realistic term projects derived from the AI domain while adhering to established SE practices.

Further details about the insightful experience gained through the implementation of this course, shedding light on the successful merging of these two fields within an educational context, can be found in the complete article published in IEEE Software [3].

## II. COURSE OVERVIEW AND DESIGN

The course aimed to allow students to develop (i) enhanced interdisciplinary problem-solving skills, (ii) improved understanding of interdisciplinary software development processes, and (iii) further career prospects with the latest trends in the market. We introduced students to the necessity for AI-intensive systems and briefly discussed the fundamentals of AI, machine learning, and statistical and data modeling. Then, the remaining lectures focused on AI systems engineering lifecycle topics, as shown by Mashkoor et al. [4]. The students

We gratefully acknowledge the LIT Secure & Correct Systems Lab (funded by the State of Upper Austria) and Austrian Science Fund (FWF) grants I4744 for supporting this work.

built their groups and came up with their proposals for the term projects, considering (i) the team comprises both AI and SE students and (ii) the project represents a real-life problem and displays the use of AI as a part of the solution.

## III. CHALLENGES AND LESSONS LEARNED

The first lesson we learned was that AI and SE are different fields requiring **different skill sets**. Integrating these skill sets in a collaborative environment can sometimes be challenging, such as differences in terminologies between the fields or differences in the nature of the resulting software, i.e., deterministic vs. probabilistic.

AI and SE students use different **tools, platforms, and infrastructure**. For example, using frameworks like TensorFlow, PyTorch, and Jupyter Notebooks is prevalent in AI. At the same time, software engineers use tools like integrated development environments, version control systems, issue tracking and management platforms, and CI/CD environments.

The SE department primarily gave the course. However, all instructors had ample experience in designing and implementing AI-intensive systems. The current course setting was adequate but not ideal. We believe **participation from AI and SE departments** is necessary for teaching such a course.

## IV. CONCLUSION

We are happy to report that all the course’s learning outcomes (LO) were achieved successfully. The student groups showed they followed the recommended practices for different lifecycle stages of systems and software engineering (LO1). During project development and its showcasing, students showed that they were familiar with AI and data science concepts (LO2), they were able to use different programming languages and frameworks effectively (LO3), and they could use SE principles for engineering AI applications (LO4). The approach of mixing students from interdisciplinary fields in a single group worked out well.

## REFERENCES

- [1] Y. Jiang, X. Li, H. Luo, S. Yin, and O. Kaynak, “Quo vadis artificial intelligence?,” *Discover Artificial Intelligence*, vol. 2, mar 2022.
- [2] A. Bewersdorff, X. Zhai, J. Roberts, and C. Nerdel, “Myths, mis- and preconceptions of artificial intelligence: A review of the literature,” *Computers and Education: Artificial Intelligence*, vol. 4, p. 100143, 2023.
- [3] A. Mashkoor, W. K. G. Assunção, and A. Egyed, “Teaching engineering of ai-intensive systems,” *IEEE Software*, vol. 41, no. 2, pp. 30–35, 2024.
- [4] A. Mashkoor, T. Menzies, A. Egyed, and R. Ramler, “Artificial intelligence and software engineering: Are we ready?,” *Computer*, vol. 55, no. 3, pp. 24–28, 2022.



# Author Index

## CSEE&T 2024

Abelson, Harold .....	131	Ghafari, Mohammad .....	191
Alibasa, Johan .....	255	Gilson, Fabian .....	51,173
AlOmar, Eman Abdullah .....	3	Groher, Iris .....	81,209
Alshehri, Yasser Ali .....	23	Hagel, Georg .....	113
Antensteiner, Tobias .....	209	Hampson, Christopher .....	273
Assunção, Wesley K. G. ....	295	Happonen, Ari .....	155
Astoto Kurniawan, Tri .....	247	Hijazi, Haneen .....	23
Azanza, Maider .....	71,261	Hingle, Ashish .....	31
Backert, Matthias .....	289	Hingle Collier, Aayushi .....	149
Bainbridge, Mike .....	273	Hofer, Alexander .....	81
Barji, Arastoo .....	267	Höhn, Ingo .....	137
Becker, Steffen .....	161,203	Huang, Wei .....	185
Bentley, Caitlin .....	273	Humayoo, Mohammad .....	185
Bigler, Dimitri .....	113	Iosif, Andrei-Cristian .....	279
Blum, Thomas .....	289	Irastorza, Arantza .....	261
Bockisch, Christoph .....	239	Islam, Md Kamrul .....	197
Böttcher, Axel .....	xxxi,119	Jachmann, Thomas .....	267
Braun, Peter .....	251	Jacobs, Sven .....	143
Breu, Ruth .....	137	Jahangirova, Gunel .....	273
Brockenbrough, Allan .....	125	Jaschke, Steffen .....	143
Brockmann, Patricia .....	247	Jesiek, Brent K. ....	149
Brozina, Stephen Cory .....	149	Jiang, Yuchao .....	241
Chan, Wing-Kwong .....	215	Johri, Aditya .....	31,149
Chen, Zhangpeng .....	241	Kaur, Harpreet .....	191
Cooling, Chris .....	237	Kegreiß, Sascha .....	xxiii
Cui, Jialin .....	13	Korte, Russell .....	149
Datta, Soma .....	235	Kovaleva, Yekaterina .....	155
DesLauriers, James .....	237	Krutz, Daniel E. ....	243,253
Dick, Steffen .....	239	Kryssanov, Victor .....	247
Doorn, Niels .....	287	Landes, Dieter .....	xxxiii
Egyed, Alexander .....	295	Lechner, Ulrike .....	227,279,293
Elhashemy, Hanya .....	131	Lee, Albert T.L. ....	215
Fischer, David .....	113	Lee, Kevin .....	113
Flatscher, Rony G. ....	xxxvii	Lee, Victor C.S. ....	215
Frankford, Eduard .....	137	Leonardos, Stefanos .....	273
Fraser, Gordon .....	91	Li, Ruochi .....	13
Galdos, Aritz .....	261	Liao, Yiwen .....	241
Galster, Matthias .....	51,173	Liu, Yang .....	243,253
Gao, Liam .....	237	López, Juan-Miguel .....	71
Garmendia, Xabier .....	71	Malachowsky, Samuel A. ....	243,253
Gasiba, Tiago (3 Papers!) .....	227,279,293	Manz, Julian .....	113
Gehringer, Edward .....	13,61	Marín, Beatriz .....	287

Marutschke, D. Moritz .....	247
Mashkoor, Atif .....	295
Mead, Nancy R. ....	xxv,101
Meawad, Fatma .....	41
Meißner, Niklas .....	161,203
Metzner, Anja .....	167
Michaeli, Tilman .....	131
Michalickova, Katerina .....	237
Minish, Matthew .....	51,173
Mirajul Islam, Muhammad .....	197
Mkaouer, Mohamed Wiem .....	3
Nair Sivasankaran, Praveena .....	155
Obermüller, Florian .....	91
Ogden, Daisy .....	273
Olmez, Muhammet Mustafa .....	61
Ongu, Didem .....	227
Ono, Yui .....	179
Ouhbi, Sofia .....	221
Paulisch, Frances .....	267,289
Pereira, Juanan .....	71,261
Pinney, John .....	237
Pinto-Albuquerque, Maria .....	227,279,293
Plösch, Reinhold .....	81
Qiu, Shuwei .....	185
Que, Xiaofan .....	243,253
Rahartomo, Argianto .....	191
Reißing, Ralf .....	245
Rössler, Andreas .....	247
Sah, Chandan Kumar .....	197
Saito, Daisuke .....	179
Salinas, Dominic .....	125
Santoso, Michael Evan .....	247
Sauerwein, Clemens .....	137,209
Simperl, Elena .....	273
Song, Yang .....	13
Speth, Sandro .....	161,203
Straubinger, Philipp .....	91
Suleiman, Basem .....	241,255
Tenbergen, Bastian .....	101,257
Thurner, Veronika .....	119
Vierhauser, Michael .....	209
Vos, Tanja E. J. ....	287
Waigner, Felix .....	247
Walker, Joanna .....	273
Wang, Anthony .....	255
Wang, Dingrong .....	243,253
Washizaki, Hironori .....	179
Wei, Zhengyuan .....	215
Wendt, Kevin D. ....	289
Xiaoli, Lian .....	197
XU, Xiaofei .....	xxvix
Yabaku, Mounika .....	221
Yin, Wenjie .....	273
Zhang, Runqiu .....	13
Zhao, Tiange .....	227,293
Zheng, Gengzhong .....	185
Zhou, Fangtong .....	13
Zhou, Xijaojia .....	185