Reflection Through Two Lenses: Experiences of Teaching and Taking Undergraduate Software Engineering and Testing Courses

Bradley Whitebread
Pennsylvania State University - Abington
Abington, Pennsylvania, USA
bew5294@psu.edu

Kseniia Gromova
Pennsylvania State University - Abington
Abington, Pennsylvania, USA
kjg5854@psu.edu

Holly Schafer
Pennsylvania State University - Abington
Abington, Pennsylvania, USA
hjs5595@psu.edu

Alok Ranjan
Holtec International Inc.
Camden, New Jersey, USA
a.ranjan@holtec.com

Ishtiaque Hussain
Pennsylvania State University - Abington
Abington, Pennsylvania, USA
ihussain@psu.edu

ABSTRACT

Undergraduate software engineering courses in higher education institutions usually cover many aspects of software development including software testing. In this experience paper, we collectively discuss our learning through realizations and reflections from both instructor's and students' perspectives of teaching and taking two separate courses in parallel, namely software engineering and software testing. Both courses had a mix of Millennial and Gen Z, junior-to-senior-level computer science students. The paper contributes by sharing the courses' design, its motivation behind the different pedagogical methodologies, tools and techniques used, and its recommendations in teaching similar courses to such a cohort of students in the future for improved learning experience and outcome.

CCS CONCEPTS

• **Social and professional topics** → **Software engineering education**.

KEYWORDS

Undergraduate Education, Millennial, Gen Z, Empirical, Experience

ACM Reference Format:
Bradley Whitebread, Kseniia Gromova, Holly Schafer, Alok Ranjan, and Ishtiaque Hussain. 2022. Reflection Through Two Lenses: Experiences of Teaching and Taking Undergraduate Software Engineering and Testing Courses. In *4th International Workshop on Software Engineering Education for the Next Generation (SEENG'22), May 17, 2022, Pittsburgh, PA, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3528231.3528354

SEENG'22, May 17, 2022, Pittsburgh, PA, USA

ACM ISBN 978-1-4503-9336-2/22/05…$15.00
https://doi.org/10.1145/3528231.3528354

1 INTRODUCTION

Software Engineering courses in the undergraduate-level usually cover many topics including requirements analysis, development best practices, design patterns, code quality, software testing, maintenance, teamwork, etc. – in an attempt to introduce students to different software engineering concepts. For the instructors, it is challenging to teach such a course since it has to combine theory and practice, especially in a short three-to-four months long semester, while allowing students to engage in a hands-on project to experience the total software development life cycle. Similarly, for students it is challenging since they need to work in a team setup (maybe for the first time), learn new techniques and tools, implement the software project using different software development tools, and meet the software requirements within the deadline.

Even though the need and importance of software testing is a well-known and established fact, unfortunately software testing, as in practice in the industry, is often neglected and does not get its due emphasis in undergraduate courses. Moreover, students also perceive software testing differently and are reluctant to learn about it [5, 6, 8]. With these challenges, how do we manage software engineering courses so that they are interesting, engaging, inclusive of both software development and testing aspects and result in a better student learning experience and outcome?

In this experience paper, we collectively share our reflections, both from the instructor and students perspective, on this overarching question. Specifically, in the Fall of 2021 at Pennsylvania State University - Abington, one of the authors (the instructor) – offered two courses in parallel, namely: CMPSC 487W: Software Engineering Design; and CMPSC 497: Special Topics on Software Testing. There were total 28 students, as shown in Figure 1, who identified themselves as either Millennial (born between 1981-1996) or Gen Z (born between 1997-2012) students (with few unknowns) and were either junior or senior year Computer Science students. The student authors of this paper were students in both courses. Moreover, for the CMPSC 497: Software Testing course, we collaborated with an industry partner who allowed the students to perform hands-on "black-box" testing on their application and find bugs. The paper contributes by sharing the courses' design, its motivation behind the different pedagogical methodologies, tools and techniques used,

and its recommendations in teaching similar courses to such a cohort of students in the future for improved learning experience and outcome. Moreover, it shares the insights of the industry partner on the steps they took, and what they would do differently the next time for such a collaboration.

In the following sections, after briefly discussing the related work, we elaborately discuss the courses' design, few teaching methodologies and the tools and techniques used, and then we candidly and in first-hand accounts describe our realizations and reflections on what went well and what could have been better to improve the overall students' learning experience and outcome.

2 RELATED WORK

The gaps between the formal teaching and training students get in higher educational institutions and what the software industry needs from the graduates is a known issue. Lethbridge [16, 17] was among the first to report about it. Honig in their paper [13] showed how the "Net generation" students (i.e. born between 1965 - 2012 since "Net generation", in this case, is an umbrella term for Generation X, Y, or Z, and Millennials) were not benefiting from the traditional software engineering education, and others [12] suggested that the fundamentals of other engineering disciplines should be included into the software engineering curricula to improve the situation. Inclusion of project-based learning, specifically team-based projects [7, 25, 27] and pair-programming [11] were some of the other improvement suggestions for software engineering courses. The importance of soft-skills development for software engineers was mentioned in several papers [18, 23, 28]. While software testing is often neglected in software engineering courses, software engineering education researchers [4, 5, 15] have urged otherwise and discussed its importance. For example, studies [10, 14, 26] have shown that test-driven development can lead to a better quality of code as well as better understanding from the students. Several studies [9, 20–23] have also found that having students work on pseudo or actual real-world projects benefits them tremendously.

Our experiences from teaching and taking the software engineering and software testing courses in parallel, as discussed in Section 4, complements and aligns with these findings. Moreover the paper contributes by sharing the details of the courses' design, pedagogical methodologies, tools and techniques they used in Section 3 and proposes actionable insights that could lead to a better learning experience and outcome for the students.

3 DESIGN, PEDAGOGICAL METHODOLOGIES AND TOOLS USED IN THE COURSES

In this section, we discuss the curricula design, teaching methodologies, and tools used in both of the courses, as well as the motivation behind using them. Few pedagogical objectives were common in both courses. However, the implementation of them may have slightly varied. The instructor laid out the courses in such a way that students will learn the theory and different concepts in software engineering and testing, as well as practice them hands-on by implementing a semester-long project by using different software development tools, as a team effort, respectively. Both courses also wanted to emphasize the importance of communication and soft skills in software development. The following subsections discuss them individually.

Figure 1: Brainstorming on the features of the project

3.1 Software Engineering Project: Improved RateMyProfessor Website with Django

Many students in North-American universities use the RateMyProfessor.com website (RMP) in reviewing their courses and instructors by providing public but anonymous feedback.[1]. They also often use it in deciding to take their future courses. However, the RMP website has some limitations: 1) as of this writing, it is only available for the UK, US, and Canadian universities, 2) it does not list course names for the reviewed courses (i.e., they use course numbers only), 3) there is no limit on how many anonymous reviews a student can post for the same course, etc. Since the students were aware of this website and the instructor faced challenges with these limitations earlier, the instructor thought this would be a great project for students to improve its features.

All the students of this class had already taken Python programming language courses but did not know the Python Django web framework [2]. To introduce Django, students were given an assignment to read and complete the official 7-parts Django tutorial [3]. in a week. Students were allowed to use the class/lecture time to sit together and complete it by helping each other. Once this was complete, students were then introduced to Git commands and GitHub.

In support of introducing remote repository concepts, the instructor, along with his lecture, used Alex Orso's lectures on Git concepts and Git commands from his course on the software development process, freely available in Udacity [2]. After these lectures, students were required to complete the interactive tutorials on "Remote Git Push Pull"[4] commands as homework. Once this training on the Django framework and Git commands were done, students were asked to create individual GitHub accounts and join the project. Next, as Figure 1 shows, they were invited to a brainstorming session to list out the possible improvement features for the RMP website and prioritize them.

Once the features were prioritized and dependencies were mapped among different features, they were divided into multiple tasks. Students were invited to bid to work on their preferred tasks. Each task was initially planned for a 2-week iteration (i.e., Sprint). Every two weeks, students were required to present their work to the class and discuss any obstacles that they were facing.

[1]The RateMyProfessor (RMP) website: https://www.ratemyprofessors.com/
[2]The Python Django web framework: https://www.djangoproject.com/
[3]The Official 7-part Django tutorial: https://docs.djangoproject.com/en/4.0/intro/
[4]The interactive git tutorials: https://learngitbranching.js.org/

Courses	Total Students	Gender		Generations			Class Year	
		Male	Female	Millennial	Gen Z	Unknown	Junior	Senior
Software Engineering	11	7	4	1	7	3	3	8
Software Testing	17	13	4	1	12	4	8	9
Overall (in %)	n = 28	71%	29%	7%	68%	25%	39%	61%

Table 1: Class demography: Total 28 students. All Computer Science majors at PSU-Abington. Majority were senior-year students.

Figure 2: Discussion on the application to test.

3.2 Software Testing Project: Perform Black-Box Testing on Industry Collaborator's System

To achieve hands-on testing experience on a real-world application, we reached out to local software industries through our professional networks and requested them to provide us with some actual application or system that we could test. After some negotiations with two such companies on the collaborations, specifically on the details of testing scope, legal obligations, working hours, and payments, we managed to partner with a multinational energy company. They would provide the necessary access and documentation to their application, and students would, in return, work part-time remotely, perform different tests, and report back their findings for free. free labor is alive and well

The application they wanted students to test was an internally developed and deployed issue reporting system for their factory floor, which did not contain any sensitive or proprietary information. They did not provide access to the source code but allowed the students to perform black-box testing on a staging environment. As part of the collaboration, before the actual testing began as shown in Figure 2, the industry partner invited the students to a field trip to their facility. They presented their application, its workflow, expected behaviors and constraints, etc. Students performed many different tests, e.g., functional, non-functional (e.g., load and performance testing), UI as well as security testings on this web application. Students performed manual testing and created an automated regression test suite for them. The outcome of this is discussed in Section 4.

3.3 Software Development and Testing Tools Used in the Courses

New hires are often introduced and required to implement and use different software development and testing tools without proper prior training. The instructor himself faced this in his own previous industry experience. To model this scenario, the instructor, after giving a brief introduction and discussing the background and usage of each of the tools, required the students to install and use them by themselves.

As part of the software engineering project implementation, students were required to use different software development tools, e.g.,: 1) for writing code they could choose integrated development environments (IDEs) such as PyCharm[5], Microsoft Visual Studio Code[6], 2) Git for version control and GitHub[7] as project repository. Students utilized the built-in code refactoring tools in those IDEs but did not use any other code formatting tools.

In the software testing course, students were introduced to different static and dynamic testing tools. Students were required to install and use the following set of testing tools, one by one, throughout the semester: Python Unit tests[8], Inductive pair-wise testing tool[9], static analysis tools, e.g., Infer[10] and FindBugs[11], load testing tool: Apache JMeter[12], PageSpeed tool to measure web core vitals[13], OWASP ZAP security testing tool[14], and Selenium WebDriver[15] for writing test automation scripts.

3.4 Industry Collaboration: Professional Perspectives from Guest Lectures

Apart from working on a meaningful, mid-sized software project and hands-on testing experience by applying different testing tools on a real-world application, the instructor wanted students in both courses to get perspectives and learn more about the culture, expectations, and daily lives of professionals, first-hand. We reached out to the software development and testing/quality assurance (QA) professionals from our networks and were fortunate to have a software development manager from a top US technology company and a former director of QA of a Fortune-500 company to come and give guest lectures virtually over Zoom. Along with technical expertise, both lecturers discussed what it takes on soft-, non-technical skills to be successful in the professional sphere and how students can prepare themselves for it. Both speakers shared some of their life

[5]PyCharm: a Python IDE by JetBrains: https://www.jetbrains.com/pycharm/
[6]Visual Studio Code, an IDE by Microsoft: https://code.visualstudio.com/
[7]GitHub repository: https://github.com/
[8]Unittest, a Python-based automated unit testing framework: https://docs.python.org/3/library/unittest.html
[9]Pairwiser, a web-based tool for pair-wise test case generation: https://inductive.no/pairwiser/
[10]Infer, a static code analysis tool by Facebook: https://fbinfer.com/
[11]FindBugs, a static code analysis tool for Java: http://findbugs.sourceforge.net/.
[12]Apache Jmeter, an open-source non-functional testing tool: https://jmeter.apache.org/.
[13]PageSpeed Insights, a tool by Google for web core metrics: https://pagespeed.web.dev/
[14]OWASP ZAP, an open-source web application security scanner: https://owasp.org/www-project-zap/
[15]Selenium WebDriver, test automation tool for websites: https://www.selenium.dev/documentation/webdriver/

lessons from their long careers and advised students accordingly during the interactive Q&A sessions.

3.5 Measures Taken for Soft Skills Development

The instructor required students to present the SOLID software design principles [19] in groups to improve the software engineering course students' communication skills, involve them in active learning of the course content, and help them work in teams. Students were randomly assigned into groups of 2-3 people; these groups, in turn, were randomly assigned one of the SOLID design principles and were tasked to prepare presentations for the class. Moreover, for each project iteration presentation, each student was required to speak in front of the class about the status of their assigned tickets, answer any questions from the other students, and help other students with their tickets; these activities targeted soft skills development.

In the software testing course, a similar scheme was adopted. Students in teams of two would need to pick, prepare slides and give a lecture explaining one of the web core vital metrics [16]. Later on, each student was asked to pick any two testing blog posts of their choice from the Google Testing Blog website [17], prepare slides, and present them in the class. A common, shared online spreadsheet was maintained so that chosen blog posts were not overlapped. This was done to ensure that a broader area of topics was covered in the class. For both presentations, students would get 8 minutes (5 minutes talk + 3 minutes Q&A) to present.

3.6 Measures Taken for Improving Written Communication Skills

In explaining the suffix 'W' for CMPSC 487W course, the official Penn State University course bulletin [18] states that it represents a second goal to help students understand the importance of written communication in software engineering and to provide opportunities for students to improve the quality of their writing, specifically in describing software systems. The bulletin further explains that the primary means of accomplishing this goal is a semester-long project in which students are required to write requirements for a large software system. In writing these requirements, students would describe the system for non-technical readers (i.e., clients and users) and specify it for technical readers (other system developers). In line with the specified written communication goal(s), the course required students to write comments and explanations for their assigned tasks/tickets in GitHub. Moreover, they were required to collaborate on writing project documentation explaining their assigned tickets, any prerequisites, and dependencies that either future developers or users might need to know to use the project. Students were encouraged to use screenshots, diagrams, etc., in explaining their work. In the end, students collaborated in a Google Document and wrote a 44-page long report.

Moreover, at the beginning of the course, for academic writing, students were given to read the recent paper on teaching software development ethics [25] and write their review for the article. Later, students evaluated each other review write-ups anonymously and discussed them collectively in a class. In the software testing course, students were assigned to read the *Software Testing Travelogue* paper by Orso et al. [24] that discusses research on software testing topics during 2000 - 2014, and watch the ACM SIGSOFT webinar on the same topic [3]. They were required to submit a document with five multiple-choice questions with answers from the paper as homework.

4 DISCUSSION

In this section, we discuss the realizations and reflections from both courses. We considered the anonymous after-course student survey feedback, 'Student Rating of Teaching Effectiveness (SRTE)', and also discussed openly about these ourselves – the main take-away points are are numbered and emphasized in the text.

4.1 Instructor-Students' Realizations and Reflections on both of the Courses

When students were required to complete the 7-part official Django tutorial prior to the project, some students were struggling to follow the instructions. Later, a step-by-step Django YouTube video tutorial [19]. helped them complete the task. A similar observation was for the testing course, when a YouTube tutorial on Selenium WebDriver [20] helped students complete their homework on creating test automation scripts. Hence, we think, *(1) Millennial and Gen Z students learn better through hands-on video tutorials, not just official documents.*

Initially students seemed reluctant to work on improving the features of RateMyProfessor.com website (RMP). After explaining the current RMP limitations and how the revised version would help students and researchers around the world to take advantage of the website, students were convinced and became interested in the project. We realized, *(2) clear motivation, goals and incentives encourage students to commit to challenges.*

Working on the project collaboratively on GitHub, even after repetitive reminders, students would not update their ticket's status with proper documentation, comments, etc. *(3) A dedicated product manager role among the students probably would have helped enforce the documentation requirements.* Moreover, after each iteration of two weeks, during the presentations, students would complain about their dependencies on others' incomplete tasks and code integration issues. Working remotely, online was worsening the problem. *(4) Dedicated in-class or in-person meetings are needed to synchronize students' work.* Moreover, few students waited till the last moment to start on their tickets, failing to complete their tasks, and jeopardizing others work. We later realized, *(5) early interventions to identify bottlenecks and removing them were crucial for a successful completion of the project.*

As discussed in Section 3.5, students were asked to give lectures on different topics in the class as part of active learning and communication skill buildup. Few students did not like the idea and complained about it in the SRTE feedback. We realized that *(6)*

[16] Google's Core Web Vitals metrics overview: https://web.dev/vitals/
[17] The Google testing blog: https://testing.googleblog.com/
[18] The Pennsylvania State University (PSU) course bulletin for Computer Science: https://bulletins.psu.edu/university-course-descriptions/undergraduate/cmpsc/
[19] The YouTube Django video tutorial: https://www.youtube.com/playlist?list=PL-J2q3Ga50oOpni_xS2PPUe4mf9lM96dD
[20] The YouTube playlist for the Selenium tutorial: https://youtube.com/playlist?list=PLzMcBGfZo4-n40rB1XaJ0ak1bemvlqumQ

discussing the motivation and objectives of class activities is necessary for students to appreciate them.

Students in software testing course had machines with different configurations (e.g., OS, memory, etc.). When students needed to install different tools into their machines to explore and use them later, it caused lots of issues and confusion. For example, to examine the static analyzer tool from Facebook, Infer, students were having issues installing it into their Windows machines, and later found out that Infer does not support Windows OS [1]. Similarly, students ran into various issues while they were installing the Apache JMeter. These difficulties discouraged students from further using and exploring these tools. This was a common theme of complaint in the after-course SRTE feedback. We recommend, *(7) using a common configuration (e.g., lab machines) in installing and experimenting with software development and testing tools for a better learning experience.*

Collaborating with an industry partner in having students test a real-world application with all the tools and techniques they learned in the class was a big success. Students spent just over eight weeks in testing their application and were able to find and report 41 bugs. The developers at the company later analyzed and reported that among the 41 bugs, there were 17 cosmetic or user interface (UI), 20 functional, 3 security and 1 performance bugs; severity wise, they marked 17 as medium to critical bugs. Figures 3 and 4 show percentage of the different reported types and severity of the bugs, respectively. The company representative thanked the class and acknowledged that these reported bugs would help them improve the quality of their system. *(8) The effort, overhead and the time spent in collaborating with an industry partner was worth the experience and learning outcome for the students.*

trusting a student's abilities + taking them seriously

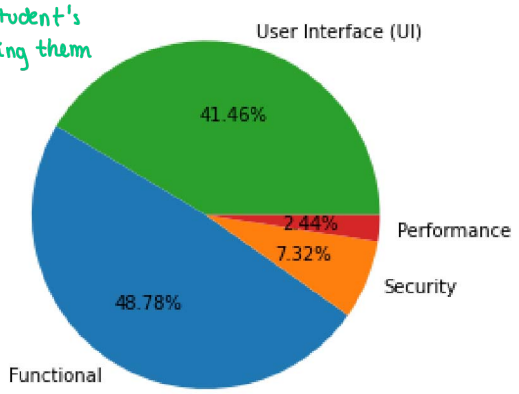


Figure 3: Types of Reported Bugs

From both courses, the main aspect that the students enjoyed and appreciated the most was the opportunity to learn and use new tools as mentioned in Section 3.3. Students grew confident with these tools as they used them throughout the semester for the projects. Students felt attached to the projects as they designed, implemented and tested them by working together in a big team while trying to achieve common goals. *(9) Different hurdles and roadblocks along the way helped them learn the importance of reading official documents of these tools and using various online resources.*

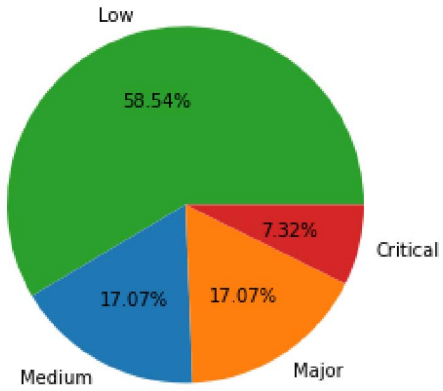


Figure 4: Severity of Reported Bugs

However, specifically for the software engineering project, students encountered two issues, poor planning on prioritization of tasks and lack of proper communication. Since some modules (e.g., data model, views) were required earlier, students responsible for those had to work harder at the beginning to complete them while others did not have much to do, and vice versa. A lack of communication between team members and communication mishaps resulted in some incomplete main functionalities of the project. These issues could have been avoided if all team members had: *(10) i) worked on the core functionalities together at the beginning, ii) added "fancy" features later and iii) had a well-defined way of communication, e.g., a group chat that should have been accessed regularly.*

Except for struggling in downloading some of the testing tools on their personal computers due to system incompatibilities and other factors, in the software testing course, *(11) overall, students were very happy for the opportunity to work with a real company in improving their system and appreciated that the course covered a wide variety of testing methods and technologies.* Finally, *(12) students benefited from giving talks on different topics in the class as this helped them polish their presentation skills, as well as allowed them to dive deeper into a topic.*

completely ignoring the feedback from those who didn't like it.

4.2 Reflections from the Industry Partner on Software Testing Project

Holtec International strives in keeping up with the latest development in academia and prides itself in giving back to the society by partnering with academia in hosting different workshops and proving students with hands-on experience on real projects through initiatives like as mentioned in this paper. When Penn State Abington faculty and students approached us, we initially thought to provide them with two applications (or app) to test. However, it was important for us to ensure that no proprietary information is exposed accidentally. *(13) A careful and through evaluation was necessary to find out all the possible data access points from the application to ensure no proprietary business process, logic or information was exposed.* After such careful evaluations, we pulled back from one app and decided to give the other app for students to perform

black-box testing on. Moreover, before we handed over the application to students to test, *(14) it was necessary for us to sanitize the application by replacing all the data access points, e.g., we replaced all the APIs and integration with other modules with mock data sets; wiped out or anonymized the database with dummy data, and hosted the application in a sandbox – completely separated from the actual application on a popular cloud platform.*

During the workshop, while demonstrating the target application, we shared with the students the end-to-end process of the application, its business logic, as well as the development process. *(15) It was important to have different stakeholders to attend the workshop, e.g., the product owner explained the business need of the application, its expected behavior and output; the architect explained how the product was designed using different design patterns and available frameworks; and the developers and test engineers explained the software development and testing process, tools, etc. This interactive session was very important for the knowledge transfer of the application.*

Students were able to understand the core functionality of the application fairly quickly and were able to find, as mentioned in Section 4, a total of 41 different types of bugs – 17 of which we categorized as medium-to-critical bugs. In our own testing, we were able to find additional bugs. All these reported and found bugs were later fixed and the application was successfully launched. In retrospect, we think we should have given the students a more detailed documentation. Students had to figure out themselves many details of the application and its workflow by performing trial-and-error based reverse engineering steps; and also a few back-and-forth email correspondence for clarifications. *(16) Providing students a detailed specification documentation beforehand would have saved valuable time and allowed them to focus more on diving deep and testing the edge cases and perhaps find more critical than lower severity bugs.*

Overall, we believe through this limited exercise, students were able to experience the real-world software testing, especially the purpose and process of software verification and validation. *(17) Again in retrospect, we believe, for the next time once we fix the reported bugs, we should plan to provide students time to further test the application to verify the fix and allow them to test more features.*

5 LIMITATIONS

We acknowledge that this paper discusses experience from two different courses. Merging all the topics, tools, and techniques from the two courses into a single software engineering course could be overwhelming for any instructor. However, we strongly believe that having authors experience both courses in parallel, both as an instructor and students, gave them a broader perspective and enabled them to share insights more thoroughly.

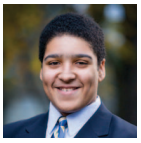
Effects of different pedagogical techniques and interventions that were used for soft- and communications skills development, etc. were not evaluated and requires further study. The experiences of the authors are subjective and cannot be generalized for all situations.

6 CONCLUSION

This experience paper discusses both instructor and students' realizations and reflections from teaching and taking two undergraduate courses in parallel, namely software engineering and software testing. It also provides the insights from the industry collaborator for the software testing project. It contributes by sharing the design, methodologies, and tools used in these courses in detail – which is reproducible. Moreover, it contributes by sharing more than a dozen learning experiences and opinions in candid, first-hand, and actionable accounts – which we believe, can benefit any software engineering and testing course instructor in offering similar courses and avoiding mistakes to ensure the maximum student learning outcome and satisfaction.

ACKNOWLEDGMENTS

We want to thank PSU Abington IT services for their help in installing tools; Gina Montgomery from the PSU Abington transportation office and Holtec International support staff for their help in coordinating and organizing the workshop. A special thanks to Anthony Zamot for connecting us with Holtec International Inc. in the first place.



Bradley Whitebread is a senior year Computer Science student at Penn State Abington. He is also currently working part-time at the startup company DeliveryCircle as a junior software engineer.

Kseniia Gromova is a senior year Computer Science student at Penn State Abington. She holds a Bachelor's degree in Education. Before switching her career, she worked as a teacher for more than seven years.

Holly Schafer is a senior year Computer Science student at Penn State Abington. She is planning to intern for Accenture in Cloud Engineering in summer 2022.

Alok Ranjan Director of Digital Innovation at Holtec International. In the past he has worked on building software solutions for several US companies: Comcast, Boeing, First Data and Lowe's.

Ishtiaque Hussain is an Assistant Professor of Computer Science at Penn State Abington. He received his Ph.D. in Computer Science from University of Texas at Arlington, and B.S. degree from University of Dhaka, Bangladesh.

REFERENCES

[1] [n. d.]. Infer does not support Windows. https://fbinfer.com/docs/support/#:~:text=Infer. Accessed: 2022-01-19.

[2] [n. d.]. Prof. Alex Orso's Software Development Process Course on Udacity. https://classroom.udacity.com/courses/ud805. Accessed: 2022-01-19.

[3] Gregg Rothermel Alessandro Orso and Willem Visser. [n. d.]. *Software Testing: A Research Travelogue*. Youtube. https://youtu.be/2t7_m68rKEU

[4] Maurício Aniche, Felienne Hermans, and Arie Van Deursen. 2019. Pragmatic software testing education. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*. 414–420.

[5] Tara Astigarraga, Eli M Dow, Christina Lara, Richard Prewitt, and Maria R Ward. 2010. The emerging role of software testing in curricula. In *2010 IEEE Transforming Engineering Education: Creating Interdisciplinary Skills for Complex Global Environments*. IEEE, 1–26.

[6] Luiz Fernando Capretz, Ali Bou Nassif, and Saad Harous. 2021. Perceptions about Software Testing among UAE Software Students. In *2021 IEEE/ACM 13th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*. IEEE, 119–120.

[7] Chung-Yang Chen and P Pete Chong. 2011. Software engineering education: A study on conducting collaborative senior project development. *Journal of systems and Software* 84, 3 (2011), 479–491.

[8] Anca Deak, Tor Stålhane, and Daniela Cruzes. 2013. Factors influencing the choice of a career in software testing among Norwegian students. *Software Engineering* 796 (2013).

[9] Nitish M Devadiga. 2017. Software engineering education: Converging with the startup industry. In *2017 IEEE 30th Conference on Software Engineering Education and Training (CSEE&T)*. IEEE, 192–196.

[10] Stephen H Edwards. 2003. Rethinking computer science education from a test-first perspective. In *Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*. 148–155.

[11] Jesús Favela and Feniosky Peña-Mora. 2001. An experience in collaborative software engineering education. *IEEE Software* 18, 2 (2001), 47–53.

[12] Carlo Ghezzi and Dino Mandrioli. 2005. The challenges of software engineering education. In *International Conference on Software Engineering*. Springer, 115–127.

[13] William L Honig. 2008. Teaching Successful" Real-World" Software Engineering to the" Net" Generation: Process and Quality Win!. In *2008 21st Conference on Software Engineering Education and Training*. IEEE, 25–32.

[14] David S. Janzen and Hossein Saiedian. 2006. Test-Driven Learning: Intrinsic Integration of Testing into the CS/SE Curriculum. *SIGCSE Bull.* 38, 1 (mar 2006), 254–258. https://doi.org/10.1145/1124706.1121419

[15] Otávio Augusto Lazzarini Lemos, Fábio Fagundes Silveira, Fabiano Cutigi Ferrari, and Alessandro Garcia. 2018. The impact of Software Testing education on code reliability: An empirical assessment. *Journal of Systems and Software* 137 (2018), 497–511.

[16] Timothy C Lethbridge. 1998. A survey of the relevance of computer science and software engineering education. In *Proceedings 11th Conference on Software Engineering Education*. IEEE, 56–66.

[17] Timothy C Lethbridge. 2000. What knowledge is important to a software professional? *Computer* 33, 5 (2000), 44–50.

[18] Henrik Hillestad Løvold, Yngve Lindsjørn, and Viktoria Stray. 2020. Forming and assessing student teams in software engineering courses. In *International Conference on Agile Software Development*. Springer, 298–306.

[19] Robert C Martin. 2000. Design principles and design patterns. *Object Mentor* 1, 34 (2000), 597.

[20] Nancy Mead, Kathy Beckman, Jimmy Lawrence, George O'Mary, Cynthia Parish, Perla Unpingco, and Hope Walker. 1999. Industry/university collaborations: different perspectives heighten mutual opportunities. *Journal of Systems and Software* 49, 2 (1999), 155–162. https://doi.org/10.1016/S0164-1212(99)00091-6

[21] Nancy R Mead. 2009. Software engineering education: How far we've come and how far we have to go. *Journal of Systems and Software* 82, 4 (2009), 571–575.

[22] Annette Miner and Brenda Nicodemus. 2021. *What Is Situated Learning?* Springer International Publishing, Cham, 17–39. https://doi.org/10.1007/978-3-030-68904-9_2

[23] Ana M Moreno, Maria-Isabel Sanchez-Segura, Fuensanta Medina-Dominguez, and Laura Carvajal. 2012. Balancing software engineering education and industrial needs. *Journal of systems and software* 85, 7 (2012), 1607–1620.

[24] Alessandro Orso and Gregg Rothermel. 2014. Software testing: a research travelogue (2000–2014). In *Future of Software Engineering Proceedings*. 117–132.

[25] Kevin Ryan. 2020. We should teach our Students what Industry doesn't want. In *2020 IEEE/ACM 42nd International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)*. IEEE, 103–106.

[26] Jaime Spacco and William Pugh. 2006. Helping students appreciate test-driven development (TDD). In *Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications*. 907–913.

[27] Anna van der Meulen and Efthimia Aivaloglou. 2021. Who does what? Work division and allocation strategies of computer science student teams. In *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)*. IEEE, 273–282.

[28] Rafał Włodarski, Jean-Rémy Falleri, and Corime Parvéry. 2021. Assessment of a hybrid software development process for student projects: a controlled experiment. In *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)*. IEEE, 289–299.