Towards Integrating Behavior-Driven Development in Mobile Development: An Experience Report

Qiang Hao
Western Washington University
Bellingham, Washington, USA
qiang.hao@wwu.edu

Ruohan Liu
Seattle University
Seattle, Washington, USA
rliu1@seattleu.edu

Abstract

Testing is an important yet often neglected skill in learning and teaching of computing science at the college level. Prior studies explored integrating *test-driven development* (TDD) into computer science courses with some degree of success, but also observed issues such as students' lack of appreciation, expressed frustration, and inconsistent adherence to TDD. TDD is a software development methodology that emphasizes writing low-level unit test cases prior to writing the corresponding portion of implementation. *Behavior-driven development* (BDD) was proposed as an evolution of TDD to emphasize software behavior from users' perspective. BDD has been widely adopted in industry, and holds great potential in addressing the issues in using TDD to improve students' learning of testing. However, BDD was rarely explored in enhancing students' mastery of testing. Informed by the literature, this experience report explored the integration of BDD into a mobile development course. Students' performance, attitude and feedback on BDD was examined, and potential improvement on the integration of BDD was discussed. The results of this report sheds light on how to effectively integrate BDD into computer science courses.

CCS Concepts

• **Social and professional topics** → **Computer science education**; **Student assessment**; *Adult education*; • **Applied computing** → **Education**.

Keywords

behavior-driven development, test-driven development, testing, mobile development, software engineering education, project-based learning, continuous integration

ACM Reference Format:
Qiang Hao and Ruohan Liu. 2025. Towards Integrating Behavior-Driven Development in Mobile Development: An Experience Report. In *Proceedings of the 56th ACM Technical Symposium on Computer Science Education V. 1 (SIGCSE TS 2025), February 26-March 1, 2025, Pittsburgh, PA, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3641554.3701875

1 Introduction

Testing is an important but often neglected skill in learning and teaching of computer science at the college level. Many studies argue that testing needs to be taught as early as possible considering its importance in authentic settings [1, 2]. However, the lack of teaching and learning of testing is still notable across computer science courses at different levels [2, 3].

TDD, as a software development methodology, emphasizes writing unit test cases before writing the corresponding functions or methods [4]. Prior efforts examining students' experience of TDD, performance on testing, and their appreciation of the testing-first approach to programming achieved some success [1, 3], but also observed challenges and issues, such as students' lack of motivation, frustration in writing trivial test cases, and lack of appreciation for the testing-first approach to programming or software development [5–7]. Interestingly, nearly all the studies on the learning and teaching of testing were limited to the entry-level programming courses.

While TDD remains a valuable approach to teaching testing in computer science courses, its emphasis on low-level code might contribute to many of the observed issues in prior studies, such as frustration in writing test cases for every method and function regardless of their triviality. Behavior-driven development (BDD), perceived as the evolution of TDD, has the potential to address the limitations of TDD. Same as TDD, BDD is a testing-first software development method. Different from TDD, BDD focuses on writing test cases that cover expected software behaviors from end users' perspective [8, 9]. Although BDD has been widely adopted in industry in the last two decades, using BDD to enhance students' mastery of testing has rarely been explored [10].

To address this gap, this experience report explored the integration of BDD into a mobile development course. This course adopted the project-based learning approach, and is powered by the usage of scaffolding, continuous integration, and generative AI. Such tools were used in concert to optimize the integration of BDD into regular learning and teaching activities. By exploring students' learning experience of BDD and testing, as well as their performance on testing, we aim to shed light on the potential of BDD in enhancing students' mastery of testing. Additionally, we hope to identify potential improvements for effective integration of BDD into computer science courses.

In the following sections, we introduce the definitions of TDD and BDD, explain their distinctions, and review their usage and potential in enhancing students' mastery of testing. After that, we describe our course design and the approaches of learning and teaching. We then report and discuss students' learning experience, performance on testing, and their attitude towards BDD. In the end, we discuss possible improvements on integrating BDD into computer science courses.

SIGCSE TS 2025, February 26-March 1, 2025, Pittsburgh, PA, USA

ACM ISBN 979-8-4007-0531-1/25/02
https://doi.org/10.1145/3641554.3701875

2 Background

2.1 Test-driven Development

Software testing is an important topic that is often neglected in undergraduate CS education. The lack of teaching and learning on testing is not only evident in entry-level CS courses, but also prevalent in upper-level CS courses because testing does not have a formal place in the existing curriculum [11, 12].

Prior studies have explored integrating test-driven development into the teaching and learning of programming to enhance students' mastery of testing. These studies observed positive effects, but also identified challenges and issues [13]. Test-driven development (TDD) is a software development methodology that emphasizes writing low-level unit test cases prior to writing the corresponding portion of implementation [4]. Prior studies on this topic have observed beneficial results on integrating TDD into programming courses, such as the studies by Buffardi and Edwards [14], Spacco and Pugh [5], and Bowyer and Hughes [15]. However, challenges and issues were also identified in teaching test-driven development, such as students' inconsistent adherence to test-driven development, lack of appreciation of and motivation to use this approach, and even frustration in attempting to achieve sufficient test coverage [5–7, 15, 16]. Beyond such challenges and issues, it is worth noting that nearly all the reported studies on enhancing students' mastery of testing were limited to entry-level programming courses, such as CS1 and CS2. Despite the importance of teaching testing to students, the efforts to enhance students' mastery of testing were scarce. Of course, *in practice* most CS curricula never tried to conform to CS2013. To our best knowledge, most studies on this topic were conducted a decade ago, and few new studies on this topic can be identified in recent years.

2.2 Behavior-driven Development

Behavior-Driven Development (BDD) is a popular software development methodology that was proposed as an evolution of TDD to address some of its perceived shortcomings. While TDD focuses on ensuring the correctness of individual units of code, BDD extends this concept by emphasizing the behavior of the system from the user's perspective [8, 9]. BDD has been widely adopted in the software industry over the last two decades [10].

BDD differs from TDD on three major aspects. First, BDD and TDD have different focuses. TDD focuses on implementation details by writing unit tests for each pieces of functionality from the developer's perspective. In contrast, BDD focuses on system behaviors from the user's perspective. BDD starts by defining the desired system behaviors using user stories and scenarios [9]. Second, BDD and TDD have different test granularity. TDD involves writing fine-grained unit tests on individual functions or methods. In contrast, BDD involves writing higher-level tests that verify the expected system behavior in terms of user interactions and business outcomes [17]. Third, BDD and TDD differ in terms of syntax. TDD sticks to programming code, whereas BDD starts by describing different usage scenarios prior to test case implementation [8, 10]. An example of BDD description is as follows:

> **Feature**: User login
> **Scenario**: Successful login with valid credentials
> *Given* the user is on the login page
> *When* the user enters a valid username and password
> *Then* the user should be redirected to the dashboard

BDD has great potential to address the challenges and issues observed when integrating TDD into programming and software engineering courses. One one hand, the testing burden of BDD is comparatively lower than TDD [10]. TDD emphasizes sufficient coverage of unit test cases on every function and method [18]. In contrast, BDD focuses on testing expected system behaviors instead of every function or method [19]. This approach can reduce the frustration of having to deal with seemingly trivial requirements to implement tests for every method regardless of their importance, making it easier for students to adhere to the testing process. On the other hand, the intrinsic value of BDD is easier to observe than TDD. The values of TDD can be relatively difficult to explain [5]. For example, is it absolutely necessary to test every function or method? Additionally, why is it unacceptable to implement first and follow up with testing immediately? Prior studies have documented the challenges in helping students appreciate the value of TDD [5, 6, 15]. In contrast, whether it is a seasoned developer or novice learner, one needs to understand the expected system behaviors before starting the implementation. That is where exactly BDD starts with, and it is natural to think about testing such system behaviors as the next step.

Given the potential of BDD, computing education community should investigate its value in enhancing students' mastery of testing. However, to the best of best knowledge, empirical studies or experience reports exploring the integration of BDD into the computer science courses are not yet identified. Additionally, the generative AI has made the teaching and learning of testing significantly easier, especially in terms of the adoption of a testing framework, and taking care of the repetition in writing test cases. Considering such major changes, BDD deserves to be explored in computer science courses at different levels, especially in software engineering courses where complex system behaviors require studying and analysis.

3 Course Design

This study aims to report an attempt to integrate BDD into a mobile development course to enhance undergraduate students' mastery of testing. This course is an advanced-level course for computer science majors. The implementation of this course followed the conventions of many reported software engineering courses, and was about full-stack mobile development. By "full-stack", it means the development of both client and server software built upon one or more frameworks, including a front-end framework and a server framework, and a database management system. In particular, this course used the following frameworks:

- *React Native*: a cross-platform framework for mobile applications
- *NodeJS and ExpressJS*: a JavaScript server framework
- *MongoDB*: a document-based database management system

The following subsections describe how this course was designed in order to integrate BDD into the learning and teaching activities.

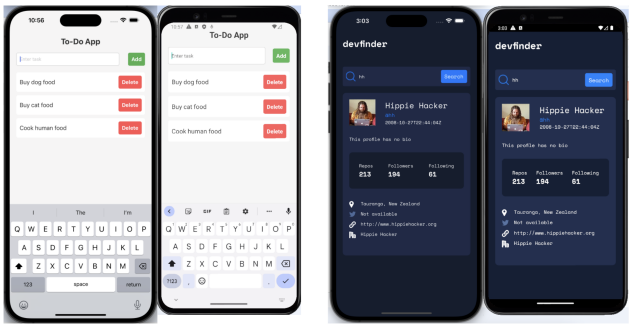

Figure 1: Sample projects covered in the lecture

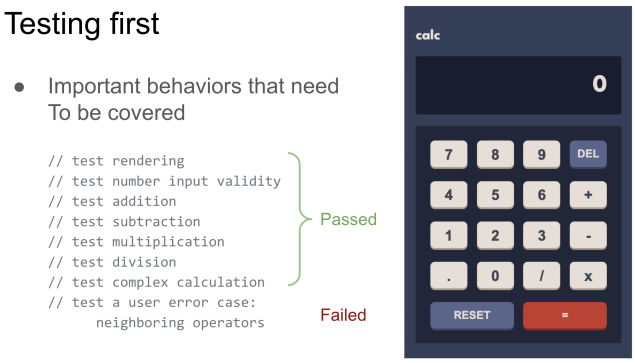

Figure 3: Testing guidelines on an assignment sample

3.1 Project-based Learning

Project-based learning (PBL) features an inquiry-based instructional approach, which engages learners in building knowledge and developing skills through the creation of meaningful projects and products based on real-life scenarios[20–22]. PBL was adopted as the main learning and teaching approach of this course. There is extensive evidence on the benefits of project-based learning on students' mastery of programming and software engineering, such as enhanced engagement and motivation, development of problem-solving skills, and exposure to real-world tools and practices [23–25].

During lectures, students were introduced to a set of different projects (see Figure 1). The instructor engaged students via live coding and interactive teaching while moving towards completing each project, so that students could pick up key knowledge components of mobile development that go beyond particular projects in this process. A key step the instructor took in this process was to stick to the principles of BDD. Specifically, each project starts with test cases covering critical expected behaviors of the mobile application to develop. *Jest*, a JavaScript testing framework, was used to structure all testing cases.

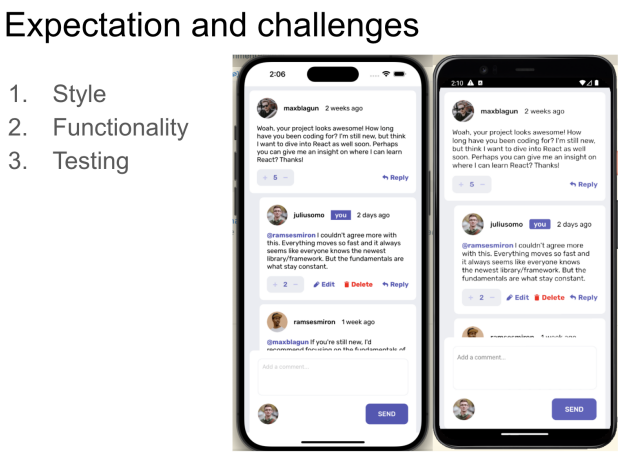

Figure 2: What students learn about an assignment in a mini-lecture

When it comes to assignments, each assignment was an independent project that requires students to design and develop a mobile application. For each assignment, students were detailed on the involved expectation and challenges on three aspects, including style, functionality, and testing via both assignment instruction and a mini-lecture that lasted 20 to 30 minutes. The following image is a snapshot of what students would learn on one of the assignments during a mini-lecture (see Figure 2). Most importantly, students were detailed on critical expected behaviors of the mobile application to be developed, and required to start their assignment by working on testing cases that cover such behaviors first. For example, Figure 3 shows an example of the testing guidelines of an assignment sample about developing a retro-styled calculator.

3.2 Scaffolding

Scaffolding was used to support integrating BDD into assignments. Scaffolding is the instructional technique of providing temporal and structured support to students as they learn and develop new skills [26–28]. The techniques assist students to complete a task or develop new understandings, so that students can later complete similar tasks alone. As students become more proficient and confident in their abilities, the support is gradually removed. Scaffolding has its unique benefits for the learning and teaching of programming and software engineering because of the incremental difficulty the subject and progressive support offered to students [27, 29]

During lectures, most projects that were covered have a "second stage" (see Figure 4 for an example). The second stage increases the project difficulty but is still based on the first stage that has been fully achieved, which lowered student cognitive load and provided opportunities for them to pick up and reinforce new knowledge components.

When it comes to assignments and the integration of BDD, scaffolding was also applied. For each assignment, students started with a provided codebase. The codebase of the first assignment has more than 2/3 of the test cases completed, and students only need to finish the remaining 1/3. The percentage of completed test cases in

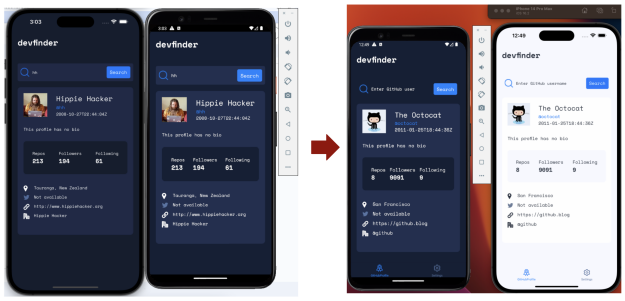

Figure 4: An example of multiple stages of the same project covered in lectures

the codebase decreased and gradually become zero when students were given the third assignment; there were 6 assignments in total.

3.3 Continuous Integration

A challenge introduced by integrating BDD into the teaching and learning activities is the extra burden of assessing students' test code. When students are asked to write test cases as a part of their assignment, they need to be assessed. The extra assessment could pose challenge for instructors, and even deter many from making testing a requirement in students' assignments.

To address this challenge, we followed the recommended approaches from Buffardi and Edwards [7, 12] to use a combination of version control and continuous integration. In the reported course, Git and GitHub were used as the version-control system to handle student assignment submissions because of their popularity in computing education and wide adoption in industry [30, 31]; GitHub Actions was used as the continuous integration tool to automate grading and provide students formative feedback. All such tools are widely adopted in industry and free for educators to use [32–34]. For every assignment of this course, a student started with a given codebase that is structured and configured to trigger the prepared test cases or the test cases students were required to develop. When a student committed and pushed his or her code to GitHub, GitHub Action would run students' code against the test cases in a virtual machine, and provide formative feedback to students via email. When the deadline of an assignment is reached, GitHub Actions can be used to facilitate or automate the grading tasks. The above process is depicted as in Figure 5.

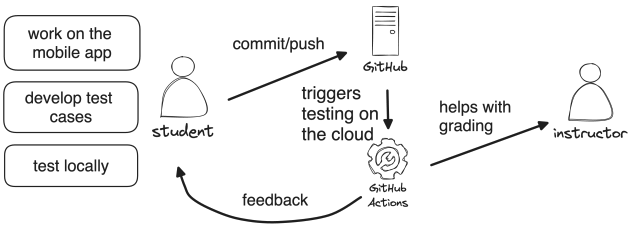

Figure 5: Workflow of assignment submission

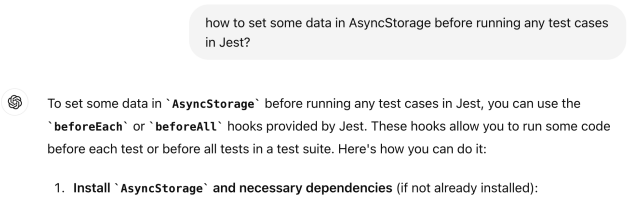

Figure 6: Using ChatGPT to answer basic questions on using Jest

3.4 Generative AI

We explored how generative AI can facilitate the integration of BDD into the learning and teaching activities. Generative AI was used throughout the course to speed up the development of test cases. The steep learning curve of how to adopt a testing framework is a common challenge that deters instructors from teaching testing in the past [35]. The rapid advancement in generative AI has the potential to make it easier to teach and learn how to adopt a new testing framework and write test cases.

During lectures, live coding was used to demonstrate the full process of adopting a testing framework and writing test cases. Generative AI was used to speed up this process. Specifically, students were shown how to combine ChatGPT and GitHub Copilot to help with writing test cases. ChatGPT was used to answer basic questions on how to use Jest (see Figure 6). GitHub Copilot was used to auto-complete test cases based on given comments.

When it comes to assignments, students were encouraged to follow what is demonstrated in lectures to use ChatGPT and GitHub Copilot to have their questions answered and speed up writing test cases.

A common concern about using generative AI in learning and teaching is that students may choose to finish the assignments using one or a few prompts without much effort, which can be detrimental to learning [36, 37]. To address this challenge, we designed each of the assignments to have complicated design specifications and functionality requirements, which are not possible to finish by relying solely on the help of generative AI. For every assignment, students were detailed on the involved expectation and challenges on three aspects, including style, functionality, and testing, so that they can have a thorough understanding of what they need to do to complete the assignments. The complexity of the assignments go beyond the capacity of generative AI. If one has to translate the requirements of such an assignment into detailed descriptions, the burden of communication will outweigh the task of completing the assignment itself.

4 Course Experience and Takeaways

4.1 Overall course experience

35 students were enrolled in this course. Out of the 35 students, 26 are male and 9 are female students. The average grade was 84.67 out of 100. Students were overall satisfied with their course experience. Their satisfaction was reflected in their answers to three key questions:

- **Q1**: How is the course overall?

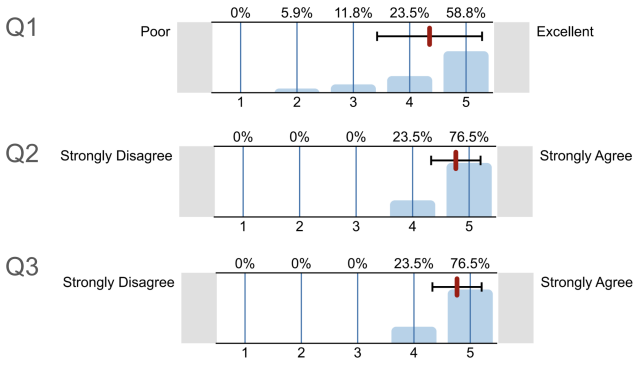

Figure 7: Students' course experience reflected in answers to three key questions

- **Q2**: Is the challenge level of this course conducive to learning and/or professional development?
- **Q3**: Do the course assignments and lectures usefully complemented each other?

Students' answers to these questions are presented in Figure 7. As is shown in the figure, students gave an average score of 4.4 to the overall course experience out of 5.0. The challenge level of the course was rated 4.6, and the tie between lectures and assignments 4.8 on average. Such scores indicate that students had an overall satisfactory experience of the course. It is vital to know that students are overall satisfactory, so that their experience on the integration of BDD won't be skewed or impacted by a negative course experience [38].

4.2 Performance on testing

Student assignments were assessed on multiple different aspects, such as design, functionality, and testing. On testing, students were assessed on whether they followed the requirements of BDD and covered all the required software behaviors in their test cases. Student performance on testing fluctuated across different assignments, but was overall solid (see Figure 8).

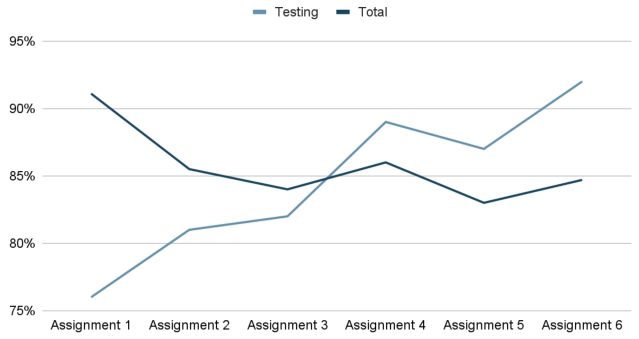

Figure 8: Students' performance on the testing aspect per assignment

The usage of Git and GitHub to manage student assignments allows us to track student progress by checking each commit and push, and determine if the requirements of BDD were strictly followed. We observed a small percentage of commits and push that violate the principle of BDD. In other words, those students chose to start developing the target mobile application without writing test cases first. However, such observation was limited in the first one or two assignments and the percentage was very low (less than 10% and 6%). As the course progressed, students all started to follow the BDD requirements without issues.

Beyond the requirements of BDD, we observed that students had little difficulty in writing test cases for the required scenarios corresponding to mobile app behaviors. Interestingly, we also observed that some students were able to come up with usage scenarios of the mobile app that are not listed in the assignment instruction, and prepare testing cases accordingly.

Overall, students' performance exceeded our expectation by a big margin. Many of the struggles that we expected students to have were not demonstrated in students' code. Students' solid performance on testing might be due to the adoption of generative AI tools, such as ChatGPT and GitHub Copilot. After all, when students have a basic understanding of Jest, they can complete most testing cases starting with a detailed comment. After that, GitHub Copilot will help auto complete the corresponding test case, and students only need to revise the parts that make less sense. Our observations are aligned with the findings of many prior studies that generative AI allows students to focus on the most important things in developing software [39–41]. In our particular case, the learning curve of adopting and using Jest, a JavaScript testing framework, is significantly alleviated. It is worth noting that mobile development is an advanced computer science course. Because of that, we did not encounter similar challenges and issues of using generative AI observed in introductory programming courses [36, 42]. The above findings propel us to reflect on the following two questions:

- How should students be evaluated on testing?
- Should students be encouraged to use generative AI to help write test cases?

In the past, the answers to these questions often depended on the course level, audience, and the goal of learning and teaching. Prior to the rapid advancement of generative AI, if the goal is for students to master the usage of a testing framework, it might be appropriate to test students' mastery of the syntax, semantics, and pragmatics of the target testing framework; if the given course is introductory, students might be required to write test cases for every function or method in their assignments [16, 17, 35]. However, instructors should anticipate for the issue of students overusing generative AI, and the challenge of ensuring fairness in grading [39, 42].

As of now, instructors who want to improve the teaching and learning of testing need to think about what the core is in terms of mastering testing. If the appreciation of testing and grasp of a workflow are more important, instructors need to think again about discouraging students from using generative AI, and how to evaluate the most important aspect of testing.

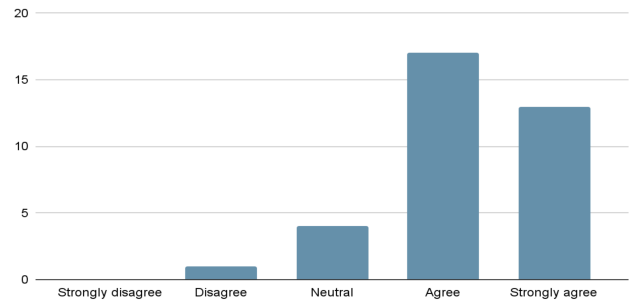

Figure 9: Students' experience in following principles of BDD

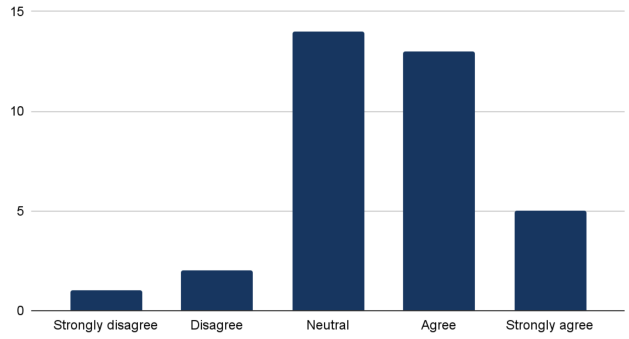

Figure 10: Students' tendency to adopt BDD in the future

4.3 Experience of learning testing and BDD

Students were asked three questions in terms of their experience of learning testing and BDD. Students' answers to the first two questions are summarized in Figure 9 and Figure 10.

- **Q1**: Did you have a good experience in following principles of behavior-driven development in writing test cases?
- **Q2**: Are you likely to stick to the principles of behavior-driven development when developing mobile applications in the future?
- **Q3**: Do you have any comments on your experience of learning testing?

Overall, students had a frustration-free experience of following the principles of BDD to write test cases first when they worked on the assignments. Additionally, students also expressed that they were likely to adopt BDD in the future to some extent. This observation is also evidenced by students' comments such as:

> *I like the testing first approach, and Copilot has made writing test cases like a breeze.*
>
> *There's a lot of testing expected in this course, but I was able to follow the requirements.*

Interestingly, we did not observe the frustration of students in writing test cases or following the testing first approach towards completing all assignments. This might be due to a combination of factors such as live coding that fully demonstrated the steps of testing, or the usage of generative AI tools. Moreover, the difference between TDD and BDD may also explain such a difference. TDD asks for test cases of every function or method, which is tedious, and sometimes trivial by nature. In contrast, BDD comes from a different angle by thinking about testing critical and expected software behaviors from the users' perspective [8, 19].

Lastly, the discrepancy between student experience of following BDD principles and their tendency to adopt BDD in the future is notable. Students rated their experience highly, and the majority agreed that they had a good experience in following the principles of BDD to write test cases. However, students showed less interest in adopting BDD in their future projects in comparison. The majority of students rated the second question (Q2) neutral. A possible explanation is that merely a good learning experience on testing and BDD is insufficient for students to fully appreciate BDD and the testing-first approach. Students might need to experience both testing-first approach and software development without testing, so that they can compare the results themselves to have a better appreciation of BDD.

4.4 Potential improvements on integrating BDD

Despite the relatively positive results, there is still a lot of opportunities to improve the effective integration of BDD into the learning and teaching activities of this course.

One notable change that we can make to enhance the integration of BDD is to slow down the pace of learning and teaching of testing. An issue that we experienced was that students felt overwhelmed at the beginning of the course because we attempted to teach too many things in a short period of time, such as how to use Jest, what BDD is, and how to plan testing of a software to build. On top of that, we released the first assignment that involved BDD almost at the same time. In the future implementation of the same course, we will adjust the pace of instruction on testing and BDD to avoid giving students a cognitive overload.

More importantly, our findings indicate that students' appreciation of BDD is hard to cultivate. A great learning experience of testing and BDD does not necessarily guarantee that students appreciate the principles of BDD, or that they would stick to it in the future [5]. Future studies may consider experimenting with failure-based education via one or two assignments that let students develop software without testing, and let students "experience the pain" of lower than expected grades when their software is buggy because of the lack of testing [43]. The contrast between sticking to BDD and otherwise may serve as a valuable education opportunity for students to see the value of testing-first approaches to software development.

5 Conclusions

This experience report explored the integration of BDD into a mobile development course. This course adopted an approach that combines the power of project-based learning, scaffolding, continuous integration and generative AI. Students expressed that they had a good overall course experience. Students' performance, attitude and feedback on BDD were examined, and potential improvement on the integration of BDD was discussed. The results of this report sheds light on how to effectively integrate BDD into computer science courses.

References

[1] Andreas Schroeder, Annabelle Klarl, Philip Mayer, and Christian Kroiß. Teaching agile software development through lab courses. In *Proceedings of the 2012 IEEE Global Engineering Education Conference (EDUCON)*, pages 1–10. IEEE, 2012.

[2] John Wrenn and Shriram Krishnamurthi. Will students write tests early without coercion? In *Proceedings of the 20th Koli Calling International Conference on Computing Education Research*, pages 1–5, 2020.

[3] Lilian Passos Scatalon, Ellen Francine Barbosa, and Rogerio Eduardo Garcia. Challenges to integrate software testing into introductory programming courses. In *2017 IEEE Frontiers in Education Conference (FIE)*, pages 1–9. Ieee, 2017.

[4] David Janzen and Hossein Saiedian. Test-driven development concepts, taxonomy, and future direction. *Computer*, 38(9):43–50, 2005.

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

[6] Mohammad Ghafari, Timm Gross, Davide Fucci, and Michael Felderer. Why research on test-driven development is inconclusive? In *Proceedings of the 14th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 1–10, 2020.

[7] Kevin Buffardi and Stephen H Edwards. Impacts of adaptive feedback on teaching test-driven development. In *Proceeding of the 44th ACM technical symposium on Computer science education*, pages 293–298, 2013.

[8] Muhammad Shoaib Farooq, Uzma Omer, Amna Ramzan, Mansoor Ahmad Rasheed, and Zabihullah Atal. Behavior driven development: A systematic literature review. *IEEE Access*, 2023.

[9] Hisham M Abushama, Hanaa Altigani Alassam, and Fatin A Elhaj. The effect of test-driven development and behavior-driven development on project success factors: A systematic literature review based study. In *2020 International Conference on Computer, Control, Electrical, and Electronics Engineering (ICCCEEE)*, pages 1–9. IEEE, 2021.

[10] Lauriane Pereira, Helen Sharp, Cleidson de Souza, Gabriel Oliveira, Sabrina Marczak, and Ricardo Bastos. Behavior-driven development benefits and challenges: reports from an industrial study. In *Proceedings of the 19th International Conference on Agile Software Development: Companion*, pages 1–4, 2018.

[11] Chetan Desai, David Janzen, and Kyle Savage. A survey of evidence for test-driven development in academia. *ACM SIGCSE Bulletin*, 40(2):97–101, 2008.

[12] Kevin Buffardi and Stephen H Edwards. Reconsidering automated feedback: A test-driven approach. In *Proceedings of the 46th ACM Technical symposium on computer science education*, pages 416–420, 2015.

[13] Matjaž Pančur and Mojca Ciglarič. Impact of test-driven development on productivity, code and tests: A controlled experiment. *Information and Software Technology*, 53(6):557–573, 2011.

[14] Kevin Buffardi and Stephen H Edwards. Exploring influences on student adherence to test-driven development. In *Proceedings of the 17th ACM annual conference on Innovation and technology in computer science education*, pages 105–110, 2012.

[15] Jon Bowyer and Janet Hughes. Assessing undergraduate experience of continuous integration and test-driven development. In *Proceedings of the 28th international conference on Software engineering*, pages 691–694, 2006.

[16] Rick Mugridge. Challenges in teaching test driven development. In *Extreme Programming and Agile Processes in Software Engineering: 4th International Conference, XP 2003 Genova, Italy, May 25–29, 2003 Proceedings 4*, pages 410–413. Springer, 2003.

[17] Avishek Sharma Dookhun and Leckraj Nagowah. Assessing the effectiveness of test-driven development and behavior-driven development in an industry setting. In *2019 International Conference on Computational Intelligence and Knowledge Economy (ICCIKE)*, pages 365–370. IEEE, 2019.

[18] Dave Astels. *Test driven development: A practical guide*. Prentice Hall Professional Technical Reference, 2003.

[19] John Ferguson Smart and Jan Molak. *BDD in Action: Behavior-driven development for the whole software lifecycle*. Simon and Schuster, 2023.

[20] Katja Brundiers and Arnim Wiek. Do we teach what we preach? an international comparison of problem-and project-based learning courses in sustainability. *Sustainability*, 5(4):1725–1746, 2013.

[21] Dimitra Kokotsaki, Victoria Menzies, and Andy Wiggins. Project-based learning: A review of the literature. *Improving schools*, 19(3):267–277, 2016.

[22] Ryan Parsons, Qiang Hao, and Lu Ding. Exploring differences in planning between students with and without prior experience in programming. In *2023 ASEE Annual Conference & Exposition*, 2023.

[23] Mehdi Jazayeri. Combining mastery learning with project-based learning in a first programming course: An experience report. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, volume 2, pages 315–318. IEEE, 2015.

[24] Jun Peng, Minhong Wang, and Demetrios Sampson. Scaffolding project-based learning of computer programming in an online learning environment. In *2017 IEEE 17th International Conference on Advanced Learning Technologies (ICALT)*, pages 315–319. IEEE, 2017.

[25] Awad A Younis, Rajshekhar Sunderraman, Mike Metzler, and Anu G Bourgeois. Case study: Using project based learning to develop parallel programing and soft skills. In *2019 IEEE international parallel and distributed processing symposium workshops (IPDPSW)*, pages 304–311. IEEE, 2019.

[26] Shu Lin, Na Meng, Dennis Kafura, and Wenxin Li. Pdl: scaffolding problem solving in programming courses. In *Proceedings of the 26th ACM Conference on Innovation and Technology in Computer Science Education V. 1*, pages 185–191, 2021.

[27] Paul Denny, James Prather, Brett A Becker, Zachary Albrecht, Dastyni Loksa, and Raymond Pettit. A closer look at metacognitive scaffolding: Solving test cases before programming. In *Proceedings of the 19th Koli Calling international conference on computing education research*, pages 1–10, 2019.

[28] Keli Luo. Navigating the code: A qualitative study of novice programmers' perceptions and utilization of automated feedback for self-regulated learning. *Education and Technology*, 1(1), 2024.

[29] Chao Mbogo, Edwin Blake, and Hussein Suleman. Design and use of static scaffolding techniques to support java programming on a mobile phone. In *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education*, pages 314–319, 2016.

[30] Qiang Hao and Michail Tsikerdekis. How automated feedback is delivered matters: Formative feedback and knowledge transfer. In *2019 IEEE Frontiers in Education Conference (FIE)*, pages 1–6. IEEE, 2019.

[31] Qiang Hao, Jack P Wilson, Camille Ottaway, Naitra Iriumi, Kai Arakawa, and David H Smith. Investigating the essential of meaningful automated formative feedback for programming assignments. In *2019 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pages 151–155. IEEE, 2019.

[32] Kai Arakawa, Qiang Hao, Wesley Deneke, Indie Cowan, Steven Wolfman, and Abigayle Peterson. Early identification of student struggles at the topic level using context-agnostic features. In *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education-Volume 1*, pages 147–153, 2022.

[33] Nils Rys-Recker and Qiang Hao. Early identification of struggling students in large computer science courses: A replication study. In *2024 IEEE 48th Annual Computers, Software, and Applications Conference (COMPSAC)*, pages 88–93. IEEE, 2024.

[34] Qiang Hao, David H Smith IV, Lu Ding, Amy Ko, Camille Ottaway, Jack Wilson, Kai H Arakawa, Alistair Turcan, Timothy Poehlman, and Tyler Greer. Towards understanding the effective design of automated formative feedback for programming assignments. *Computer Science Education*, 32(1):105–127, 2022.

[35] David Janzen and Hossein Saiedian. Test-driven learning in early programming courses. In *Proceedings of the 39th SIGCSE technical symposium on Computer science education*, pages 532–536, 2008.

[36] Brett A Becker, Michelle Craig, Paul Denny, Hieke Keuning, Natalie Kiesler, Juho Leinonen, Andrew Luxton-Reilly, JAMES PRATHER, and KEITH QUILLE. Generative ai in introductory programming, 2023.

[37] James Prather, Brent Reeves, Juho Leinonen, Stephen MacNeil, Arisoa S Randrianasolo, Brett Becker, Bailey Kimmel, Jared Wright, and Ben Briggs. The widening gap: The benefits and harms of generative ai for novice programmers. *arXiv preprint arXiv:2405.17739*, 2024.

[38] David H Smith IV, Qiang Hao, Filip Jagodzinski, Yan Liu, and Vishal Gupta. Quantifying the effects of prior knowledge in entry-level programming courses. In *Proceedings of the ACM Conference on Global Computing Education*, pages 30–36, 2019.

[39] Paul Denny, Juho Leinonen, James Prather, Andrew Luxton-Reilly, Thezyrie Amarouche, Brett A Becker, and Brent N Reeves. Prompt problems: A new programming exercise for the generative ai era. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1*, pages 296–302, 2024.

[40] Beiqi Zhang, Peng Liang, Xiyu Zhou, Aakash Ahmad, and Muhammad Waseem. Practices and challenges of using github copilot: An empirical study. *arXiv preprint arXiv:2303.08733*, 2023.

[41] Ben Puryear and Gina Sprint. Github copilot in the classroom: learning to code with ai assistance. *Journal of Computing Sciences in Colleges*, 38(1):37–47, 2022.

[42] Joyce Mahon, Brian Mac Namee, and Brett A Becker. Guidelines for the evolving role of generative ai in introductory programming based on emerging practice. In *Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1*, pages 10–16. 2024.

[43] Andrew A Tawfik, Hui Rong, and Ikseon Choi. Failing to learn: towards a unified design approach for failure-based learning. *Educational technology research and development*, 63:975–994, 2015.