Comps Paper

Alec Phillips
aphillips2@oxy.edu
Occidental College

1 Problem Context

Software testing is a core principle of computer science that is often overlooked or at least under-emphasized in computer science education. Testing is an integral part of any software project, and is a skill that must be developed like any other technique in software development. It takes considerable knowledge to be able to develop robust and efficient test cases that create confidence in one's code. Thus, this skill should be honed and developed along with other programming techniques. Software testing is also nuanced and there are numerous testing techniques that are important for students to learn about and have experience putting into practice. The importance of software testing serves as the motivation behind my comps project idea; I would like to create a web application that teahches software testing techniques and exposes introductory to intermediate level students to related concepts.

A web application that teaches software testing would be beneficial to computer science education in general because it could be added to a standard computer science curriculum to help students get exposure to the types of testing and how they are implemented. My goal is to provide comprehensive materials on software testing as well as hands on exercises where learners can put the concepts into practice. Instructors could add this to their curricula, or self-learners could utilize it to gain exposure to software testing techniques.

2 Technical Background

This section will cover the relevant terminology that is critical for understanding the technical side of my project as well as the general goals of my application. These include general software testing techniques, as well as related topics such as test-driven development, error handling, edge case identification, and debugging.

2.1 Software Testing Techniques

2.1.1 Scopes of Testing

Testing is a nuanced aspect of software development, and there are several key techniques that are commonly employed for writing tests. Software testing can be broken up into four main categories depending on the scope and motivation of the test. These categories are unit testing, integration testing, end-to-end testing, and acceptance testing [9]. These are the four categories of testing that I plan to teach in my application.

- Unit tests are those that have the narrowest scope, applying to individual modules of code. Defining a specific unit of the code is up to the developer, but it is standard for unit tests to apply to specific functions, although they can also apply to an entire module.
- Integration testing checks that the individual units are interacting as expected. This means that the interfaces between the units as well as the general information flow between them should be exercised.
- System, or end-to-end testing evaluates the overall requirements of the entire application. System tests would generally involve providing inputs at the most general or external level, and evaluating that the outputs are as expected. For these tests to pass, all the internal processes of the aspect being tested must be functioning correctly.
- Acceptance testing is the final step, in which it is determined if the piece of software conforms to the general specification requirements, and also aligns with what the client or user is expecting [9].

These different techniques are often used together to comprehensively exercise code, as they work together to test a single application in different ways. Thus, it is not sufficient to know just one of these techniques; a competent developer should be familiar with all of them, in order to know what to utilize depending on their needs.

2.1.2 Arrange, Act, Assert

Along with these scopes of testing, my application includes information on testing 'best practices'. These include the *arrange*, *act*, *assert* style of writing tests. This is the generally accepted format of structuring each test case. Each test case should include each of the following steps, which should be clearly differentiable.

1. The *arrange* step involves setting up any context necessary to run the test - this could include initializing objects and variables that will be involved in the test.

2. In the *act* step, function calls to and interactions with the code being tested are made. This is where the *behavior under test* is exercised. Results of these interactions are saved so that they can be verified in the *assert* step.
3. The variables resulting from the *act* step are then checked for correctness in the *assert* step. This involves some form of *assertion* statement that will see if the act step resulted in the expected values and throw an error if the result is not as expected.

This format is followed because it makes tests easy to read. Tests should be straightforward so that when something breaks in the code, it is easy to determine where the error arose by looking at the failing test cases. This is why consistency in formatting and test style is important.

2.1.3 Code Coverage

Code coverage is an important metric to be familiar with for evaluating test quality. Coverage refers to the percentage of code that is exercised by a series of tests. There are several types of coverage, including statement, branch, and path coverage. Statement coverage refers to the percentage of lines of code (not including empty lines) that get run. Branch coverage refers to the percentage of logical branches of the code that are run - this means that each branch of an if/else statement constitutes a different branch. Path coverage is the most robust, referring to the percentage of possible paths through the code that get exercises. Thus, for full path coverage to be met, each combination of branches would need to be tried by a different test case. My application uses branch coverage as the metric to determine test quality of users' exercise submissions.

2.2 Test-Driven Development (TDD)

Test-driven development is not a testing strategy, such as the ones discussed in the prior section. Instead, it is a software development technique, meaning that it informs the entire software development process, not just the testing aspect [7]. In general, TDD is a strategy of software development that prioritizes testing and shapes the development process around the writing of tests. This is an important concept for students to be familiar with because it is a common practice in industry.

Initially, I had wanted my application to specifically have students practice the process of test-driven development. However, after beginning to develop coding exercises, I decided that it would be more engaging for students to be able to immediately run their test cases against a specific function. Thus, I did not include exercises specifically on test-driven development, but it is still explained in the Learn section of my application.

Although my application did not end up revolving around Test-Driven Development, it was important in motivating the project because of TDD's popularity and usefullness as a development technique.

Steps of TDD:

1. Write test cases for the next unit of code being added to the project
2. Write the code that the first test case is exercising (until it satisfies the test case)
3. Refactor the code as needed (both function and test case)
4. Re-run all existing test cases to check for regressions in the code base [2]

This process then repeats until the project is complete, thus informing the development of the entire project, not just the testing aspect.

3 Prior Work

This section explores some of the literature on incorporating software testing into computer science education, as well as some of the areas that this integration can be improved. The following research guided and motivated my project, helping me determine what my application should focus on and the gaps that it should aim to fill.

3.1 Software Testing in Education

This section will discuss some of the strategies that have been used to teach software testing and test-driven development in the past. This will inform the approach that I take for teaching software testing in my application. I came across a significant amount of discussion on the reasons why software testing is undertaught, as well as why students find it to be an uninteresting concept to learn. Additionally, there was discussion of how test-driven development could be integrated into computer science education.

The lack of emphasis on software testing in education stems from both students and instructors. Computer science courses as well as entire programs are already packed with material, so adding lessons or assignments focused solely on testing can be infeasible [5]. Additionally, students often find implementation of projects more exciting than testing their code, and testing can appear tedious. Students can also be unmotivated to test their code, because they do not want to see it fail [3]. This parallels what I have noticed as a teaching assistant; students often write test cases that are specific to their implementation that they know will pass. I have also noticed that students generally do not take writing test cases very seriously. Even when instructed to write tests, students in Data Structures will treat it as an afterthought, heavily prioritizing other aspects of their projects.

If the incentives were flipped, and students had more of a desire to write failing test cases, they would likely take the task more seriously. This informed the way that I designed my practice exercises in the application; instead of having students write code and then test it, I provide them with code that they then need to exercise with test cases, and their performance is evaluated by how robust their tests are. Ideally, this incentivises the user to thoroughly understand the code they are testing and creates a desire to write good tests that cause the code to fail.

There is also existing research on applying the concepts of test-driven development in general computer science courses. This would make software testing a more integral part of every project that students take on; one strategy for implementing this in an educational setting was to make students fully responsible for demonstrating the correctness of their code [4]. This means students would not be given any automated test results before submitting their code, and would instead be responsible for determining their code's correctness on their own. If this strategy were to be implemented in a classroom setting, it would be even more important for students to have a solid baseline understanding of testing. Although my application does not prompt students to work in this way, this research still motivated my project design.

4 Ethical Considerations

An application geared towards education may initially appear to be ethical, however there are still potential moral issues. Any application or product geared towards users that can affect real world outcomes may cause negative change. Additionally, there are a number of ethical concerns that arise from taking on the role of educator; for instance: what are the ethical and pedagogical obligations that an educator has to the students? There is the also the topic of power, as this application's service will likely only affect those who already have access to computer science education, reinforcing the current power dynamics in terms of who has access to education and the ability to learn computer science. Additionally, it reinforces power dynamics existing around who has internet access. Despite the initially benign appearance of this project, these ethical considerations emphasize that there are still potential ethical issues present that need to be considered carefully.

4.1 Educator Obligations

Taking on the role of educator places one under an ethical burden, as they are entrusted to have expertise on the topic and the ability to adequately convey accurate information. The University of Michigan education department offers a simple overview of nine core ethical obligations that teachers should uphold. Several of the obligations that they bring up will be difficult for me to uphold within the context of this project, specifically personal responsibility and competence.

The webpage defines personal responsibility as taking "responsibility for obstacles to student success and to work assiduously to ensure equitable access to learning opportunities" [10]. This poses an issue for my project because of the amount of time I will be able to dedicate to maintaining this project after it is completed and the impersonal nature of an educational application. If this application ends up actually getting deployed, it would be infeasible for me to be responsive to every user and their individual needs.

The article defines competence as "[developing] and continually [working] to improve instructional competence, and to strive to engage in professionally-justified teaching practice at all times" [10]. This is similarly infeasible because of the commitment that it would take to actively continue improving my instructional competence. Within the context of my project, it would require continued updating of the materials and content, as well as continuing to further my knowledge to better serve the users of the application, which are not responsibilities that I can confidently commit to at this time. These two examples represent the issues that come along with taking on the role of educator and building an educational site. In order to address this, it is important that, if this application is to be deployed, I only allow it to be active for as long as I can uphold these obligations.

4.2 Power Reinforcement

In addition to the general ethical issues of building an education based application, there is the problem of who will have access to this application, and how that reinforces existing power dynamics in the world of computer science and technology. Power and computer science are tightly intertwined; technology is nearly omnipresent in our world, and many of the most influential and powerful individuals are those who control large technology companies. One clear factor that differentiates power in computer science and technology is whether or not someone has access to the internet. Individuals who have easy access to the internet are much more easily able to interact with existing technologies and web applications, and also have much more access to learning how to interact with computers in general. The issue with this web application, and conveying educational material over the internet in general, is that it is only making information more accessible to those who can already easily access it. Even if my application successfully makes the information easier to learn or more enjoyable, it is not reaching a new audience or further distributing access to learning computer science. This then reinforces the existing power structures that exist in computer science.

There is already a large power difference between those who have access to the internet and those who do not. The more effective my application is, the more I would be contributing to that power difference. The power differences caused by unequal internet access have been well documented, and play a role in global power dynamics as well. The advent of the internet has created easy access to information and communication, however this only applies to those who have access to it. Societies that do not have the luxury of widespread internet access do not get these same benefits, which increases the power differentials between these societies [6]. This emphasizes the importance of the relationship between internet access and power. Therefore, further empowering those with internet access by means of a web application that only those individuals can benefit from will inherently further these existing power dynamics.

Power differences as a result of internet access have been further exacerbated by the COVID-19 pandemic. This has been a prevalent issue in the academic sphere; students who have access to reliable internet can more easily and consistently access their courses, and do not have to worry about missing material because of something outside of their control like stable internet [8]. This emphasizes that, currently, those without internet access are experiencing even greater challenges as a result of the pandemic, so introducing more web based applications will further this divide.

These issues are important to consider, but my hope is that ultimately my application can provide more benefit to individuals and make computer science more accesible overall. Additionally, the way that my application is designed works to mitigate these concerns. Specifically, because the application can be fully contained within the user's web browser and there are no network calls made during use, the application actually does not require an internet connection. If being used in the browser, a network connection is only required when the user initially requests the page. However, the application code can also be downloaded onto the device and can then be completely used with no internet connection. This makes the application more accessible and helps to combat some of the concerns above regarding the furthering of the power divide caused by internet access.

5 Methods

5.1 Implementation

This section will discuss high level implementation decisions and their rationale. For more detail on techniques used, see the code documentation within the project-code/README file.

5.1.1 Serverless Design

The project is build fully using React.js and javascript, along with CSS and HTML. Originally, I intended to have a server-side API to process code submissions, and a database to store the excercises and user progress. However, this would require the application to have a login page, and the code evaluation mechanism would involve subprocesses or something analogous to asynchronously run code submissions. Additionally, running user generated code on the server-side is inherently dangerous, because this code gains access to the file system and OS of the server. In order to safely run their code, I would have needed to either restrict what the user is allowed to write in the code editor, or heavily verify their submissions. This would be very difficult and time consuming to do sufficiently, and would take away from the ability to focus on other aspects of the application like usability.

All of these factors motivated me to build the application as a server-less web-page. This way, the user generated code is only running in their web-browser, and so a user has no ability to break a remote server with their code. Additionally, there is no reliance on a network connection to submit code, and no need to handle multiple submissions concurrently, as would be necessary if a back-end were implemented. This makes the application very fast, since it is independent of the speed of the network. One drawback of this design is that all coding exercises have to be in javascript, because that is the only language that can run in the web-browser, without the use of some sort of cross-compiler. I had initially wanted to have the exercises be in python, but javascript is also a good language for students to get introduced to.

One main drawback that the serverless design posed was the lack of a database. My main concern without a database was figuring out how to store the content descriptions, exercises, and user progress. However, I realized that another way to achieve this was through the browser's localStorage feature. This is a structure stored by the browser that each webpage can access through a simple get/set API. I leveraged this to store a list of exerciseIds corresponding to the exercises that the user has completed. Upon a correct exercise submission, the application updates this list, which is saved and loaded on subsequent visits to the application. In order to get around storing other data that would have been put into a database, I stored the data in javascript objects. This simulates the way that the data would be stored in a non-relational database. I looked into serverless database options, but after learning about localStorage, I chose to go down that route because it seemed more feasible for the scale of the application.

Additionally, the serverless design made the application much easier to deploy; I was able to release through github pages, which was a very simple process. This made it easier to have an initial prototype out earlier, meaning I was able to do a first round of user evaluations fairly quickly, allowing me more time to iterate on feedback and get input from users.

5.1.2 Extracting Code Coverage

Another aspect of the implementation that required outside research was getting code coverage for a user's test cases. From an initial search online, there were many tools that gather test coverage, but few that talk about how test coverage is actually determined. One technique that I located showed the process of injecting coverage flags into a function under test, which can be set to true when the execution enters the corresponding condition [1]. To implement this I simply declare an array of boolean flags, all set to false, with each flag corresponding to a particular branch of the code. When the function runs, within each branch, a different element of the array is set to true. Once all tests have been run, the total branch coverage is the percentage of flags that are set to true. This was a surprisingly simple way to determining coverage, and was effective for my purposes. However, path coverage would have been a more thorough metric to use for evaluating user test quality. I would have liked to extend to detecting path coverage with more time to work on the project, because it seems interesting algorithmically, and would require users to write more thorough test cases.

5.2 Interface Design

My main motivation with the interface was to have a simple but visually appealing design that focuses more on an intuitive page layout. My general design was inspired by CodingBat, which is one of my favorite online computer science education resources [11]. This site was created by a Stanford professor for his introductory computer science students. I find the overall experience of using the website to be intuitive and simple. The user interface is bare-bones and easy to navigate. The mixture of written content and coding exercises was particularly influential in my application design.

One difference was that I wanted my application to have a more clear organization between learning and practice materials. This informed my decision to create a clear split between the two areas. I had initially wanted to pair learn and exercise sections, but after putting together all the materials, I realized that there would be more learn sections that excercise sections. This ultimately motivated me to split the two areas.

5.3 Content Design

As an educational application, a critical piece of my project was designing the content and programming exercises. I made sure to talk to many individuals about what would be most helpful to include in the application. The main takeaways were that it is important to include hands-on exercises as well as detailed examples, as opposed to simply having descriptions on types of testing. However, it was also emphasized that the materials should be concise to avoid being overwhelming. This was important to me as well, because I want this to be a concise and useful introduction to the nuances and types of software testing. One helpful piece of advice that I recieved on designing the exercises was to focus on common errors, such as off by one and improper input formats.

In terms of the specific types of exercises included in the application, I had initially wanted to focus on implementing a combination of unit and system-style testing exercises. However, some aspects of development that I had overlooked took more time than expected, and I ended up focusing more on including a combination of edge-case identification and unit testing exercises, with minimal system test exercises. This ended up being more feasible within the time constraints of the semester. However, I believe that the application still holds value as a teaching tool or for general student use. The edge-case identification exercises are still beneficial because it teaches students to reason about code correctness and consider the range of possible inputs and deeply understand the code presented. Additionally, these exercises prompt users to both identify an input that causes a function to break, and then actually debug the code. This is useful for gaining a deeper ability to reason about code, and offers an entry point into learning javascript for the user.

The test-writing exercises involve writing test cases that use assertion statements, with the goal of maximizing branch coverage on a given function. In early problems, the code being tested is provided, so the user can step through the function and more easily determine what inputs to provide to fully cover the branches. However, in subsequent exercises, only a description of the behavior of the function is provided, with the actual code obscured. This forces the user to consider how the function would be implemented and the necessary control flow. To aid the user in this task, on a given submission attempt, they are given feedback on the percentage of branches that their code covered, as well as whether any of their assertion statements failed.

I ended up implementing one system-level test exercise, which involves the user writing a series of assertion tests on a larger scale API. This exercise is the most difficult of all the content and is only unlocked after a user has completed all the unit test exercises. This way they have the proper context to be able to complete the system test exercise. This problem involves writing test cases against a mock course registration system. Documentation for a CourseScheduler class with its member method stubs is provided. Additionally, users are given the starting state of a 'database' which is actually implemented as a series of javacsript objects. This simulates the experience of writing code that interacts with a back-end database. For this exercise, users are also trying to raise the overall branch coverage, and to do so must interact with all the methods in the CourseScheduler class and reason about what combinations of interactions will cause certain operations to occur. For instance, students need to exercise a prerequisite-checking function, but to do so they must enroll students in courses, increment the semester (so that students have then completed prerequisite courses), and then enroll the students in courses they are now qualified for.

5.4 Iterative Project Improvement

Since I was able to deploy an early version of the project, I had time to do two rounds of user evaluations, allowing me to make iterative improvements to my project based on user feedback. In my first round of feedback, I used a survey that gauged the quality of the teaching content, the clarity of the exercise directions, the difficulty of several of the exercises, as well as overall user experience. From this first round of evaluations, I learned that the directions for the exercises could be made more clear, and that it would be helpful to include hints for some of the more difficult exercises. Additionally, it was noted that the feedback on coding exercises could be more detailed so that the user has more of a sense of what they did wrong. After this feedback, I added a hint area for each exercise, and improved the feedback reporting area to be more clear and visually appealing.

In addition to using a survey, I watched individuals interact with the application to get a sense of how clear it is to use and where individuals get confused. One main takeaway from this was that users sometimes found it difficult to know where to start with the coding exercises. The exercises are displayed in rows, and I intended for them to be attempted from top-left to bottom-right. However, I noticed that this was not always intuitive, and this informed my decision to add the feature of unlocking exercises as the user progresses. This enforces the order and provides a clear visual of where to start.

Interactive testing also exposed some bugs in the code evaluation system, specifically with the system test exercise. One more advanced student noticed that a specific series of interactions caused the exercise to break, so I was able to identify and fix the bug. Additionally, users identified multiple spelling errors in the application.

Overall, the iteration on feedback allowed the usability of the application to be improved and gave me confidence that the adjustments being made were rooted in creating a better experience for real users.

6 Evaluation Metrics

To formally evaluate my project, I chose to focus on three main criteria:

1. Teaching Quality (of written teaching content)
2. Exercise Quality (clarity of directions, difficulty)
3. Overall Learning Outcomes

As discussed above, these were evaluated in two stages, first on a larger scale with a survey that was sent out to members of the computer science community, and secondly with one-on-one user interviews where interactions with the application were observed. The ideal evaluation method would have been to implement the application in one section of an introductory course, and compare those students to a control group who were not exposed to it. However, with the time constraints this was not feasible. Additionally, although the user survey was not my preferred form of evaluation, it proved to be effective for hitting the three main criteria listed above.

6.1 Round 1 - User Survey

6.1.1 Survey Design

The survey was designed to target the three areas above for testing. Specifically, users were prompted to:

- Read the content on unit testing
- Read the descriptions of three exercises (one edge case identification, one debugging, one unit testing)
- Attempt to complete the three problems within two to three minutes

This allowed me to gain a sense of how effectively each of the evaluation criteria were met in an early version of the project. At this point in the implementation there were not yet system test exercises, so this aspect could not be evaluated. The problems that users were asked to complete were the rainfall problem for edge case and unit testing, and a fizzbuzz variation with an off by one error for debugging.

6.1.2 Participation

The survey was sent to ten upper division computer science students (students either majoring or minoring in computer science who have participated in an upper division course), as well as one section of twenty-eight introductory computer science students. Seven of the upper division students participated in the survey, and no introductory students participated. This was problematic because my application was intended to be geared towards introductory students. However, the findings actually indicated that the more advanced students could still benefit from using the application.

6.2 Round 2 - One-on-one Evaluations

The second round of user evaluations involved asking individuals about their experience using the application in a one-on-one setting. These types of interviews were conducted with six students, three advanced and three introductory level. I used two different formats for this.

- Watching individual users interact with the application and observing with minimal interference
- Asking individuals to use the application on their own (without observing them) and then discussing their experience

This was intended to gain insight into how users interact with the application with no constraints or specific guidelines. The decision to observe some users and not others was motivated by the fact that some testers seemed to look for guidance when I was observing them, which defeats the purpose of the user interacting with the application freely. However, it was still beneficial to get a direct visual of how individuals navigate the page. Additionally, the non-observational type allowed me to get feedback from individuals who were not directly present.

With each of these tests, my evaluation consisted of an open ended discussion with the user about their experience, as well as more targeted questions about their interactions. These questions focused on the three main evaluation criteria detailed above, so as to be consistent and work towards building a sense of the key goals of my project. These conversations also focused more on the overall usability of the application, and the clarity of the general application flow.

7 Results and Discussion

7.1 Round One Evaluation Results

The raw data for this evaluation can be found in the ./comps-paper directory of the project.

7.1.1 Teaching Quality

For the category of teaching quality, students indicated that their prior knowledge of unit testing was ∼2.14 on a scale from one to five. After reading the 'Learn' section on unit testing, respondents rated their understanding at a 4 on the same one to five scale. Additionally, they rated the clarity of the section as ∼4.57 out of five. This was a satisfactory outcome, indicating that even more advanced students felt that the section taught them something new. This is especially telling because I would have expected advanced students to be most familiar with unit testing.

7.1.2 Exercise Quality

To evaluate exercise quality, for each of the three problems attempted, students indicated the clarity of the exercise description and whether they were able to solve the problem. Repondents reported average clarities of ∼3.86, 4, and ∼4.23 out of five for the edge case, debugging, and unit testing exercise directions respectively. These ratings indicate that users were generally able to determine what was expected for each type of exercise. However, in the written feedback area, some users indicated that the input format for edge case problems could be made more clear. In terms of difficulty, the solve rates were 71.43% (edge case), 85.71% (debugging), and 42.86% (unit test). I would have rated these as easy to medium difficulty in terms of the problems that could have been chosen. These results indicate that students had the most trouble solving the unit testing exercise within the time limit, which reinforces the general idea that students would benefit from such exercises/practice. This also gave me a good baseline for understanding how my perception of problem difficulty translates to users' experiences.

7.1.3 Learning Outcomes

In terms of overall learning outcomes, it becomes clear from the findings above that students seem to benefit from interacting with the application. This is emphasized by the fact that even upper division students felt that they gained knowledge from the written materials, and also were not able to easily complete the unit testing exercise on the rainfall problem. Additionally, on a Likert scale from one to five, when asked if they would like this application to have been incorporated into their introductory computer science course:

- 14.29% gave a rating of 3
- 57.14% gave a rating of 4
- 28.57% gave a rating of 5

Overall the sentiment was positive and indicates that students would generally like to see this incorporated into their curriculum. Also, 100% of respondents indicated that they learned something through interacting with the application.

Despite the limited engagement with the survey, especially from introductory students, the findings were informative in providing a baseline of the applications effectiveness in relation to the three main evaluation criteria.

7.2 Round Two Evaluation Results

There is no raw data for this round of evaluations because they were done in an informal/casual way through conversations with users.

The one-on-one evaluations were particularly insightful for indicating how usable the application is, and whether it reached my goals for intuitiveness and clarity. Some main takeaways from watching individuals interact with the application were that introductory users tended to start with the 'Learn' section, while more advanced students wanted to start with the exercises, specifically the harder ones, instead of working their way up through the problems. In these cases, I would manually unlock the more advanced levels for them, but they tended to struggle because they were missing context that would have been explained if they had started from the easier problems. This reinforced my idea that it makes sense to still enforce the ordering of the problems even for more advanced users. Beginners also had difficulty, but were still able to solve the easier problems after reading the directions and contextual material. It was encouraging to see that students of all skill levels found the problems to be adequately difficult.

In terms of usability, testers indicated that the content sections were clear and that they were not confused about how to navigate the application. Users seemed to want to work from left to right through the content. There were still some points of confusion, specifically on the submission feedback for the coding problems. For the edge case identification problems, some users were confused because the goal of the problem is to provide an input that causes the function to execute incorrectly, but when they provide an input that successfully does this, the feedback says 'PASS.' Users found this confusing because they thought that this meant the function executed correctly, not that their input broke the function. I tried to fix this problem by using more visual cues that their solution passed, such as highlighting the submission green. Also, a green check-mark will appear on the problem name and the next exercise becomes unlocked. Given more time, I would further improve this by forcing the page to scroll back to the top or providing a 'next exercise' button to make it even more clear that the problem is complete. Another point of confusion was that, for coding problems, the feedback indicates if there is an error in the user's code, but it does not say which line the error is on. Users complained that this made it difficult to debug their code. I tried to address this, but it seems like a limitation of the way the evaluation mechanism works; the user's code is embedded into an anonymous function that actually gets run in a virtual machine within the browser. I do not know all the specifics of why this is the case, but it seems to be the same for dynamically generated functions and any code run with the eval() function. I would also like to have tried other ways of addressing this if there was more time to work on the project.

7.3 Conclusion

Despite some of the shortcomings discussed, I believe that the project succeeded in meeting the goals that I set out to accomplish when designing the project. My main goal was that users would get a learning experience from using the application and be exposed to new content in a manageable and understandable way. From both sets of evaluations, it became clear that not only introductory, but also advanced students benefitted from interacting with the application. I believe that this indicates that the concept has more utility than I had initially thought. This also shows that more advanced students do not always learn these concepts independently when not taught these ideas in an introductory course. I believe that this reinforces the importance of introducing students to these concepts. In addition to the improvements discussed above, the main change that would be necessary if this application were to be used in a course is that more content would need to be added. It definitely proved to be more time consuming than anticipated to come up with exercises of appropriate difficulty - often what seemed like a good idea for an exercise proved to be significantly too easy or challenging to solve. It was specifically difficult to come up with unit test exercises, because the complexity of an algorithm often did not translate to having a significant number of branches in the logic, meaning that it would only require a couple test cases to fully reach the coverage goal. In conclusion, the current state of the project serves as a valid educational tool, with infrastructure in place that could be easily extended to become even more robust and effective.

References

[1] Baxter, Ira D. *Branch Coverage for Arbitrary Languages Made Easy*. Tech. rep. Austin, TX: Semantic Designs, Inc., 2001.

[2] Bhat, Thirumalesh and Nagappan, Nachiappan. "Evaluating the Efficacy of Test-Driven Development: Industrial Case Studies". In: ISESE'06. Rio de Janeiro, Brazil: ACM, 2006.

[3] Carrington, David. "Teaching Software Testing". In: ACSE'97. Melbourne, Australia: ACM, 1997.

[4] Edwards, Stephen H. "Rethinking Computer Science Education from a Test-first Perspective". In: OOPSLA. Anaheim, CA: ACM, 2003.

[5] Edwards, Stephen H. "Teaching software testing: automatic grading meets test-first coding". In: OOPSLA. Anaheim, CA: ACM, 2003.

[6] Fang, Mei Lan et al. "Exploring Privilege in the Digital Divide: Implications for Theory, Policy, and Practice". In: *The Gerontologist* 59.1 (May 2018), e1–e15.

[7] George, Boby and Williams, Laurie. "A structured experiment of test-driven development". In: *Information and Software Technology* 46 (2004), pp. 337–342.

[8] Lai, John and Widmar, Nicole O. "Revisiting the Digital Divide in the COVID-19 Era". In: *Applied Economic Perspectives and Policy* 43.1 (2020), pp. 458–464.

[9] Luo, Lu. *Software Testing Techniques: Technology Maturation and Research Strategy*. Tech. rep. Class Report for 17-939A. Pittsburgh, PA: Institute for Software Research International, Carnegie Mellon University, 2001.

[10] Michigan School of Education, University of. *Ethical Obligations*. 2022. URL: `https://soe.umich.edu/academics-admissions/degrees/bachelors-certification/undergraduate-elementary-teacher-education/ethical-obligations`.

[11] Parlante, Nick. *CodingBat: code practice*. 2017. URL: `https://codingbat.com/java`.