CS 320 Project 2

# **Summary**

1. **Describe your unit testing approach for each of the three features.**
   1. **To what extent was your approach aligned to the software requirements? Support your claims with specific evidence.**

The software testing techniques that I have implemented for the Task Service, Contact Service, and Appointment Service within Project 1 included JUnit tests and coverage tests. When I designed and developed my code for the task service assignment, I followed the software requirements exactly. For each individual requirement I made sure that the proper functionality was there designed and implemented within my code. To make sure that the functionality worked properly within my code, I utilized multiple Junit tests to make sure I had 100% test coverage. Some of the software requirements required multiple Junit tests to make sure all the different methods worked properly but the proper functionality was there. So, you could say that my testing and the software requirements aligned very closely together throughout my development process.

* 1. **Defend the overall quality of your JUnit tests. In other words, how do you know your JUnit tests were effective based on the coverage percentage?**

Throughout the testing process of writing JUnit tests for the task service assignment, I would periodically check the JUnit coverage for each of the 6 files within the application (Task, TaskService, Appointment, AppointmentService, Contact, ContactService). There were some methods that I implemented that either had a couple conditional statement or multiple return statements and I initially looked over some of this and it initially lowered my coverage percentage. Thankfully with the use of the JUnit coverage test I just kept writing tests until I eventually achieved 100% coverage within all the application files excluding the JUnit test files. In terms of quality of coverage, having 100% test coverage is as good as it gets but this typically does not give the full picture of the overall quality of the tests and there are other important factors to consider as well when writing high quality tests.

1. **Describe your experience writing the JUnit tests.**
   1. **How did you ensure that your code was technically sound? Cite specific lines of code from your tests to illustrate.**

I made sure that my code was technically sound by ensuring that it meets all the software requirements while also utilizing modern and efficient coding practices during the development process. I also used JUnit tests within my code to make sure that I had 100% coverage within my source code, and although this won’t always find all the potential bugs, it is a great method to make sure my application has all the necessary functionality. I will attach a screenshot of some of the JUnit tests I

implemented to ensure the creation of the various objects in my application worked properly.

Text

Description automatically generated

Figure Appointment Object Test

* 1. Text

     Description automatically generatedText

     Description automatically generated**How did you ensure that your code was efficient? Cite specific lines of code from your tests to illustrate.**

Figure Task Object Test

Figure Contact Object Test

Text

Description automatically generatedI made sure my code was efficient in many ways with the first being my use of object orient programming. Object oriented programming allows my code to be more efficient through inheritance and encapsulation features which can make my code more reusable/maintainable. Specific lines of code within my project that illustrate OOP and encapsulation that include all my accessors and mutators throughout the Task(lines 31-56), Contact(lines 38-78), and Appointment(lines 32-40) objects.

Figure OOP and Encapsulation Example for the Appointment Object

Text

Description automatically generatedText

Description automatically generatedAnother way that I tried to keep my code efficient was by utilizing an array list to store all my various objects within my application. With the use of my array list, I can very easily search and delete various objects within the list with a given unique ID for the specific object. If this project were to significantly increase the number of contacts, I would end up switching to a different data structure such as a hash table or binary search tree but for the scope of this class I believe that a standard array list is efficient. I attached a screenshot of the specific lines of code where I implemented my search and delete methods within the TaskService, ContactService, and AppointmentService files.

Figure OOP and Encapsulation Example for the Contact Object

Figure OOP and Encapsulation Example for the Task Object

Text

Description automatically generatedText

Description automatically generatedText

Description automatically generated

Figure Storing the Contact Object in an Array Example

Figure Storing the Appointment Object in an Array Example

Figure Storing the Task Object in an Array Example

# **Reflection**

1. **Testing Techniques**
   1. **What were the software testing techniques that you employed in this project? Describe their characteristics using specific details.**

The software testing techniques that I have implemented for the application within project 1 included JUnit tests and coverage tests. JUnit is a popular open-source framework that is used for creating and running automated tests in Java. It enables developers to write test code that validates the intended functionality of their code, detects any errors and exceptions, and handles anticipated and unanticipated inputs. JUnit supplies developers with a collection of annotations and APIs, which facilitate the formation of test cases and grouping them into test suites. When developers write tests for individual or groups of methods, JUnit executes the tests in a specific sequence and produces a report containing the results. JUnit coverage testing was the other method that I frequently used throughout the milestones. Coverage testing is a testing method that assesses the degree to which a software application's source code is tested. This form of testing calculates the code coverage of automated tests, that is, the proportion of source code that is executed during testing. Code coverage is a metric that quantifies the percentage of code lines or statements that the tests exercise. It gives developers an understanding of how extensively their code is being evaluated and helps pinpoint any sections of the code that require additional testing.

* 1. **What are the other software testing techniques that you did not use for this project? Describe their characteristics using specific details.**

There are various other testing techniques that are used but I will only go over a couple due to the scope of this journal. The first technique I will go over is integration testing. This technique tests the interactions between the different components of the software application to ensure they are working together properly. System testing is another commonly used technique with its purpose being to ensure the entire system meets the set requirements and performs as expected. Performance testing is another technique that could be implemented to check the software applications’ performance under various stressful conditions to ensure it can keep up with all the set requirements. Lastly, I could implement security testing to make sure the software applications existing security is able to protect against data theft, unauthorized access, and various other security attacks.

* 1. **For each of the techniques you discussed, explain the practical uses and implications for different software development projects and situations.**

Integration Testing: Integration testing is a software development technique used to verify that various components or modules of a system function correctly and as intended when tested together. This type of testing is crucial in complex software applications that comprise numerous interconnected modules and systems.

System Testing: System testing is a software testing technique that is carried out after integration testing and before user acceptance testing. This type of testing aims to verify the functionality, performance, reliability, security, and other essential attributes of the software application to ensure that it meets the specified requirements and performs as expected. System testing is especially crucial in the case of large-scale software applications that involve several components, subsystems, and interfaces. The testing helps to confirm that the application works accurately as a whole, and any defects or issues are identified and resolved before the final release.

Performance Testing: Performance testing is typically carried out after functional testing and before system testing, usually in the later stages of the software development life cycle. Its primary objective is to identify any performance bottlenecks and other issues that may have an impact on the application's performance and user experience. Performance testing is critical for web-based applications and software that deal with large volumes of data or user traffic. The testing ensures that the application can handle the expected workload and provides optimal user experience under various scenarios.

Security Testing: Security testing is typically performed in the later stages of the software development life cycle, after functional testing and before the application is released to the market. It comprises various testing techniques, including penetration testing, vulnerability scanning, and threat modeling, to evaluate the application's security posture. Security testing is especially crucial for software applications that manage sensitive or confidential information, such as financial data, personal details, and medical records. It assists in detecting and addressing security vulnerabilities before the application goes live, minimizing the risk of data breaches and cyber-attacks.

1. **Mindset**
   1. **Assess the mindset that you adopted working on this project. In acting as a software tester, to what extent did you employ caution? Why was it important to appreciate the complexity and interrelationships of the code you were testing? Provide specific examples to illustrate your claims.**

In this specific project I wouldn’t say that I employed much ‘caution’. This was because of the scope of this assignment being to learn and explore various coding and testing methods, I did try some new things/techniques such as JUnit testing and coverage testing. As far as appreciating the complexity and interrelationships of the code, I would say that this is extremely important for me because it can help identify potential issues and improve the overall quality of my code being developed. In almost all coding projects there are multiple components that are connected within code and when a change is made in one place it is often affected in multiple other areas as well. By thoroughly understanding these relationships, I as a developer can create more efficient and quality JUnit tests to ensure the application is functioning as intended. Testers who comprehend the relationships between different code components can provide developers with more precise bug reports and detailed feedback. This can result in quicker bug resolution and an overall improvement in the quality of the software.

* 1. **Assess the ways you tried to limit bias in your review of the code. On the software developer side, can you imagine that bias would be a concern if you were responsible for testing your own code? Provide specific examples to illustrate your claims.**

When a software developer is responsible for testing their own code, bias can be a potential concern. This is because developers may have a personal or professional interest in the success of their code, leading them to overlook or downplay any biases or issues that may exist. Moreover, developers may have a limited perspective on how their code may be used by others or how it may interact with other systems, which can create blind spots in their testing approach. To address this concern, it is crucial to have a testing process that involves multiple parties, such as developers, testers, and stakeholders. Developers should collaborate with testers to identify potential biases or assumptions in their code and work together to create a comprehensive testing plan. Furthermore, organizations can implement testing policies and procedures to ensure that all code is thoroughly tested and validated before release. Some examples of bias include confirmation bias, anchoring bias, availability bias, similarity bias, and groupthink bias. It is crucial for developers to be aware of these biases and take steps to mitigate them. By recognizing and challenging their own assumptions and beliefs, developers can improve the quality and effectiveness of their work, leading to better software products for users.

* 1. **Finally, evaluate the importance of being disciplined in your commitment to quality as a software engineering professional. Why is it important not to cut corners when it comes to writing or testing code? How do you plan to avoid technical debt as a practitioner in the field? Provide specific examples to illustrate your claims.**

Cutting corners in writing or testing code can have adverse effects on software quality, reliability, and maintainability. It is essential to avoid shortcuts to ensure that the software meets the expected standards. Skipping essential steps, such as proper testing, code reviews, and documentation, can lead to bugs, errors, and crashes that may result in dissatisfied customers and lost revenue. Moreover, cutting corners can make the software harder to maintain. If the code is poorly structured or not documented, it becomes challenging for other developers to understand and make changes to it, leading to longer development cycles, higher maintenance costs, and a higher risk of introducing new bugs. This can also lead to a decrease in the reliability of the software. If testing is not done thoroughly, it can lead to unexpected failures in production, causing downtime, lost revenue, and reputational damage. Cutting corners can also lead to technical debt, which is the cost of maintaining and updating the software over time due to shortcuts taken during development. Technical debt can accumulate over time and become a significant burden for development teams, making it difficult to deliver new features or improvements. Because of this, here are my methods to avoid as much technical debt as possible during the development process:

* Writing maintainable code: Writing code that is easy to read and understand can help ensure that it is maintainable in the long term. This can involve using clear variable names, writing comments, and organizing code into reusable functions and classes.
* Testing early and often: Testing code early and often can help catch errors before they become problematic. This can involve using automated unit tests and integration tests.
* Refactoring code: Refactoring code regularly can help ensure that it remains maintainable and scalable. This can involve breaking large functions into smaller ones, simplifying complex code, and removing duplicate code.