### Southern New Hampshire University

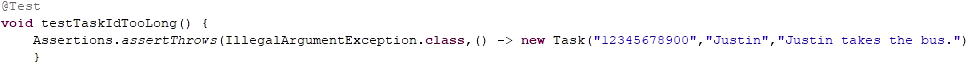
### 7-2 Project

Chris Wills

### Instructor Omar Toledo Lopez

### CS-320-R1876 Software Test Automation & QA 23EW1

10/15/2023

1. **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. | My approach to the project and each category of code was solely aligned to the software requirements received. In this case, my grade received would have been altered negatively if I hadn’t done so, but in the real world the client would have likely brought up issues that would have impacted the project either financially or with increased time to complete a finished product. Found in my code will be illegal argument setups to ensure the required field meets the requirements set out by the client. In my JUnit tests, you will find tests that will create instances of invalid information trying to be entered in to ensure the code responds appropriately. Take these two instances for example: In this instance the Task ID is required to be no more than 10 characters. The initial exception is created when creating the Task ID variable. Then the JUnit test on the second section of code tests this exception by entering in an ID that contains more than 10 characters.
      2. 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?  **|** Out of all of my JUnit tests, the lowest percentage of coverage came out to 87.2%. The general rule for a reasonable amount of coverage is 80% (Garcia, 2017). Most of my tests hovered around close to 90% coverage, with one even reaching 100% in my TaskServiceTest. These high coverage percentages show that my JUnit tests implemented were focused on achieving high quality of testing.
   2. 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. | By utilizing testing methods that test most if not all of the methods created in the corresponding class help ensure the code being tested has been successfully implemented, thus technically sound. A computer code with text

         Description automatically generatedFor example, in the test code segment above the Delete Task method is being tested with trying to delete a task that does not exist. We first add the task and test the list to see if the task has been populated into the task list. We then test the delete method with an invalid Task ID which should show an exception since that particular ID does not exist. After that, we test the list once again to ensure the list hasn’t actually removed any items as it wasn’t meant to. Success across this test verified the class code created is functioning correctly.
      2. How did you ensure that your code was **efficient**? Cite specific lines of code from your tests to illustrate. **|** Having compartmentalized code in the test helps increase efficiency as methods can be called for without having to utilize extraneous methods that could slow down processing. A screenshot of a computer code

         Description automatically generatedThe code segments above, for example, are different test methods testing to delete a contact and updating a contact’s first name. By containing these tests in self-contained and well labeled methods, dependency can be reduced or eliminated between the two. One of these can be removed without affecting the other.
2. **Reflection**
   1. Testing Techniques
      1. What were the **software testing techniques** that you employed in this project? Describe their characteristics using specific details. **|** Since this project was the culmination of the milestones we had completed previously, the same testing methodologies were used in the final coding project. I utilized both structured-base (white box) and specification-based (black box) testing techniques in the creation and revision of the code submitted. As white box testing derives test cases directly from the structure of a component or system, structured-based testing entails breaking down individual code statements, segmenting tests, and evaluating three crucial coverages: branch, statement, and path coverage (Hambling et al., 2019). The purpose of specification-based testing techniques is to demonstrate how the system model will behave as expected within the program. These techniques include clearly differentiating between valid and invalid user inputs, using decision tables to test various conditions and the actions that result from them, and using state transition tests to assess the effects of events that cause changes and produce outputs. By employing this strategy, it is possible to create test scenarios that are based on particular use cases and guarantee thorough testing of boundary values, ensuring the fulfillment of all requirements.
      2. What are the **other software testing techniques** that you did not use for this project? Describe their characteristics using specific details. **|** In the project still, I did not utilize experience-based testing methods. Problem-solving in error resolution frequently involves intuition and exploratory testing. Error guessing aids in identifying specific tests for system analysis, while exploratory testing targets areas needing additional specifications. These techniques rely on user and tester expertise to systematically evaluate different parts of the system, ensuring consistent and error-free usage.
      3. For each of the techniques you discussed, explain the **practical uses and implications** for different software development projects and situations. | The choice between these techniques depends on the specific needs of the project, the level of access to internal code, and the desired focus of the testing efforts. In conclusion, structured-based testing is advantageous for in-depth code analysis, security testing, and complex systems, while specification-based testing excels in user experience evaluation, integration testing, and requirements validation. As long as time constraints permit it, using all of the mentioned testing methods helps deliver thorough and comprehensive testing of future projects.
   2. 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. **|** As this was my first foray into software testing, caution was employed heavily as I was developing the classes and corresponding tests. Developing and creating too much and too advanced of code would have impacted my ability to test the code in a negative way, as it would have likely required more complicated and lengthy tests. It is vitally important to maintain constant knowledge and understanding of how the code comes together between the different classes and tests, as without this constant appreciation I would have likely found myself frustrated and at a loss on what fixes to implement that were causing errors in different areas of the overall project. One time I did not employ caution and created a more complex working environment was when I was implementing a randomly generated ID for some of the tests instead of allowing the user to create their own. This changed some of the testing methods I developed that wouldn’t have covered an actual user test case scenario.
      2. 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. **|** People often can grow attached to things and projects they create, and this can lead to bias when reviewing said project. While it was gratifying for me to have built up this code, taking a step back and changing my mindset from a proud creator to a less biased reviewer and inspector was necessary. Confirmation bias might be more readily present, especially when dealing with code built by one’s self. Software developers and testers are more likely to select positive tests than negative ones due to the phenomenon known as confirmation bias, which is defined as the tendency to verify one's own hypotheses rather than trying to refute them (July, 2013). In my experience in limiting my bias, using the testing coverage tool in Eclipse assisted tremendously. A lower test coverage indicated I may have been biased in my initial test creation, either in unconsciously thinking my code was already correct and ready, or in having an unconscious blind spot to a particular method of code.
      3. 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. **|** Avoiding technical debt and having in depth testing in place that covers as much test cases as possible results in potential money and time being saved, as well as increasing the customer satisfaction and quality concerning the product. Not to mention testing will often help in the security of the product, reducing the chance of catastrophic data leaks impacting the client and consumers. Releasing untested or poorly tested code is the same as releasing unfinished code, as testing is a necessary step in the development lifecycle. In the example of this project, had I not gone back and tested my code with thorough JUnit testing, I would have missed out on some critical errors, particularly with the Contact class and its phone number variable. My phone number variable did not have the correct length or character type set (numerical digits) that could have resulted in invalid responses being accepted. I would have released a product that did not meet the requirements set out by this course or the client. Quality assurance teams and operations prevent future complaints, streamline development, and create an overall higher-functioning software for consumers. Software testing is essential in developing and launching a well-loved, lucrative product (Verma, 2023).

**References**

Garcia, B. (2017). *Mastering Software Testing with JUnit 5*. Packt Publishing.

Hambling, B., Morgan, P., Samaroo, A., Thompson, G., & Williams, P. (2019). *Software testing: An ISTQB-BCS Certified Tester Foundation Guide*. BCS, The Chartered Institute for IT.

July, G. M. (2013, July 18). *Bias and the human side of software testing*. SmartBear.com. https://smartbear.com/blog/bias-and-the-human-side-of-software-testing/

Verma, M. (2023, February 7). Don&#039;t Cut Corners: 5 Reasons Software Testing Matters. *ePRNews*. https://eprnews.com/dont-cut-corners-5-reasons-software-testing-matters-483273/