Project Two

**To what extent was your approach aligned to the software requirements? Support your claims with specific evidence.**

For each of the modules, my implemented tests were written to follow the rubric as close as possible so as to not skip over any requirements. For developing the actual tests, I had a goal in mind to keep each section of code within the tests to test for a single function. This made troubleshooting very easy since if one the tests failed, I would only have to chase one issue due to each test only testing for simple functionality. This mindset helped me to ensure all tests passed without fail and met all requirements. I have provided a snippet of code from the project to show how I made tests only test for one piece of functionality, rather than multiple so the code can be as modular as possible:

@Test

public void testUpdateTask() {

TaskService service = new TaskService();

Task task = new Task("1234567890", "Valid Name", "Valid Description");

service.addTask(task);

service.updateTask("1234567890", "New Valid Name", "New Valid Description");

}

@Test(expected = IllegalArgumentException.class)

public void testUpdateNonExistentTask() {

TaskService service = new TaskService();

service.updateTask("1234567890", "New Name", "New Description");

}

**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?**

I made sure each test had one hundred percent coverage to meet all requirements. I made each test in regards to the grading rubric and prompts. I know the tests were effective as I tested small pieces of functionality per test so any errors could be singled out. This helped me identify if the test itself was an issue or if I was missing functionality.

**How did you ensure that your code was technically sound? Cite specific lines of code from your tests to illustrate.**

I deployed the use of junit tests to verify that all of my code functioned as needed. Each tested made sure that code met requirements and functioned as intended. I have provided an example of my code along with its associated junit test showing how I tested the updateTask function as well as trying to update a task that did not exist:

TaskService.java

public void updateTask(String taskId, String name, String description) {

for(Task t: tasks) {

if (t.getTaskId().equals(taskId)) {

t.setName(name);

t.setDescription(description);

return;

}

}

throw new IllegalArgumentException("Task ID: " + taskId + " does not exist. Please try again.");

}

}

TaskServiceTest.java

@Test

public void testUpdateTask() {

TaskService service = new TaskService();

Task task = new Task("1234567890", "Valid Name", "Valid Description");

service.addTask(task);

service.updateTask("1234567890", "New Valid Name", "New Valid Description");

}

@Test(expected = IllegalArgumentException.class)

public void testUpdateNonExistentTask() {

TaskService service = new TaskService();

service.updateTask("1234567890", "New Name", "New Description");

}

}

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

I ensure efficiency within my code by having a modular mindset during the milestones. I made each section modular so code could be updated, removed, or added very easily. Each section within the services accomplish a singular task rather than multiple things at once. Testing multiple things at once would lead to less and smaller amounts of code but this approach is not very good for a modular approach as multiple processes are then tied together. Here is an example of where I did provide an efficient way to check for multiple requirements as the requirements were tied together and there was no point in testing these requirements separately:

public Appointment(String appointmentID, Date appointmentDate, String description) {

**if (appointmentID == null || appointmentID.length() > 10)** {

throw new IllegalArgumentException("The appointment ID cannot be blank or longer than 10 characters.");

}

**if (appointmentDate == null || appointmentDate.before(new Date()))** {

throw new IllegalArgumentException("The appointment date cannot be blank or in the past.");

}

**if (description == null || description.length() > 50)** {

throw new IllegalArgumentException("The description cannot be blank or longer than 50 characters.");

}

this.appointmentID = appointmentID;

this.appointmentDate = appointmentDate;

this.description = description;

}

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

For this project, I used Junit tests to verify that my code aligned with the requirements to function as needed. The junit tests were straightforward and easy to design as each of my tests only tested for small amounts of functionality. This made the tests modular and easy to understand from a non-technical standpoint. The self-verifying nature of tests made them extremely quick to run.

**What are the other software testing techniques that you did not use for this project and explain the practical uses and implications for different software development projects and situations.Describe their characteristics using specific details.**

A type of testing that I did not use for this project was end-to-end testing. These types of tests verify that all components contained within a system function as required. I did not have to implement this type of testing as we were not designing and building an entire system. These tests are usually deployed as a comprehensive type of test to verify functionality at the end of development. I also did not use performance testing. Performance testing would have been used to validate that the program performs within expectations. When this is combined with stress testing, an entire system’s bottlenecks can be identified and fixed to improve the overall product along with the user experience.

**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.**

During the development of the different milestones, I was cautious in the way that I developed my tests. I attempted to think of any other requirements that may have been implied based off other requirements. One time that comes to mind was during the task service milestone. The requirement for the code was to implement functionality that allowed for the program to update an already existing task. I was cautious and provided a test that attempts to update a nonexistent task even though the rubric did not call for it as it is something a user could attempt to do.

**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.**

It is very hard to test the code that you wrote without having some kind of bias. I found that the best way for me to cut-down on bias was to write the tests on a sperate day rather than the day that I finished the tasks themselves. I would go back and look at just the requirements as information for my test development rather than write the tests based off the code I had written. By not looking at my code until the tests were completed, I could write the tests in a way that did provide any type of leniency.

**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.**

It is our job as software developers to provide code that is only helpful to humanity. Providing code that is free from defects with no malicious intent is what we should strive for every day. Some of this software controls extremely important parts of our lives such as navigation systems within an airplane, software that handles financial information in areas such as banks, or even things as obvious as the computer being used to type this. If thorough testing is not conducted before releasing a product, it could put many people’s livelihoods or even lives at stake. Every piece of code developed should be tested beyond what is required to ensure that corners are not cut to deliver products with defects and flaws.