Shawn Millin

CS-320

Final Project Two

December 10th 2024

Summary and Reflection

**Summary**

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

Junit testing was aligned with software requirements by following the requirement’s specifications exactly for both contact object, task object, and appointment object. Both objects had similar requirements for their variables with all of them requiring not to be null, had to be under a certain length, and not able to update. Updating variables came into the ContactService, TaskService, and AppointmentService java file’s coding for certain variables.

Example of following requirements of length and not null throwing an error if input is out of that scope:

**if**(name == **null** || name.length()>20 ){

**throw** **new** IllegalArgumentException("Unacceptable Name");

Example of updating a variable’s assignment:

**public** **void** updateAddress(String updatedString, String contactId) {

**for** (**int** i = 0; i < contacts.size(); i++) {

**if** (contacts.get(i).getContactId().equals(contactId)) {

contacts.get(i).setAddress(updatedString);

**break**;

}

**if** (i == contacts.size() - 1) {

System.out.println("Contact Id not found.");

}

}

}

**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?**

By running a Junit test through the test files a resulting chart showed the passing, error, and failing codes. Running a coverage Juint test all test functions were run and a percentage chart showed the percentage of code that passed the test of throwing an error for each

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

Technically sound by making each variable distinct from another, separating methods for adding, and deleting tasks or contacts. Variables were set to private to not let other programs without permission access these variables. Made use of the getter and setter methods for setting updatable variables and getting the assigned values for each variable as well as the use of a constructor for each class.

Example of variables and constructor:

**public** **class** Appointment {

**private** String appID;

**private** Date appDate;

**private** String appDesc;

**public** Appointment(String appID, Date appDate, String appDesc){

///(code omitted for length reasons)

}

Example of getter:

**public** Contact getContact(String contactId) {

Contact contact = **new** Contact(**null**,**null**, **null**, **null**, **null**);

**for** (**int** i = 0; i < contacts.size(); i++) {

**if** (contacts.get(i).getContactId().contentEquals(contactId)) {

contact = contacts.get(i);

}

}

**return** contact;

**}**

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

Efficiency in the code is in the simplicity of the code. This code was not filled in with unnecessary added code to over complicate or clutter the program.

Examples of efficiency:

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

**throw** **new** IllegalArgumentException("Invalid ID");

}

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

**throw** **new** IllegalArgumentException("Invalid Date");

}

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

**throw** **new** IllegalArgumentException("Invalid Description");

}

and:

**public** **void** addTask(String taskId, String name, String description) {

Task task = **new** Task(taskId, name, description);

tasks.add(task);

**}**

**Reflection**

Testing Techniques

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

All three milestones completed for this class have used Static Analysis in the testing of the code. Syntax errors were checked through the use of the Eclipse IDE. Manual Testing technique was also done by manually going through the code to find any errors that the IDE did not pick up on such as spelling mistakes or mistakes in fulfilling the requirements.

Unit Testing technique was used in testing each java file by using the Junit Test. For this an import of assumptions and import of the java test libraries was done to test each of the requirements in the java files to test the functions of each method. The tests were done to test the boundary of the requirements such as an input being too long, too short, a past date, or null.

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

Dynamic Testing was not used in the testing of the three milestones for no code was actual ran.

Integration Testing technique also did not take place in the milestone testing for each unit was tested separately, combining each unit in the future would take on the Integration Test to test how well each unit is integrated with the next. Non-functional testing, Performance testing, and Security Testing techniques also did not come up in the milestone testing. Security testing would take place if the program was going out to a customer or being sent out for public use. Performance testing would be used if there was a need to test for speed, scalability, and stability.

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

Of the techniques discussed the Manual Testing technique is not for every project. Large and complex projects have too much code to manually sift through so Manual Testing is for smaller projects and project where there is enough staff and time to do so. All projects that are going to public use will need Security Testing and Performance Testing to both increase user interaction and keep out infiltration or security breaches. Dynamic Testing is also necessary for programs going to a customer or for public use because the test can catch certain interactions that were not caught in the initial testing of the program’s code.

**Mindset**

**Assess the mindset that you adopted working on this project. In acting as a software tester, to what extent did you employ **caution**?**

To make sure each test was accurate I wrote and tested each test before writing the next test. This allowed me to pinpoint any faults or errors that may have occurred instead of writing all test at once and testing all at once, which would have made finding an error more difficult.

**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?**

Bias could occur in testing when a developer tests their own code because they may not have the catch code that could be written slightly different than what they are used to or they missed a requirement.

**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?**

Cutting corners could lead to not fully meeting requirements, a lack in security measures, and could lead to additional work later on to fix errors in the code.

**How do you plan to avoid technical debt as a practitioner in the field? Provide specific examples to illustrate your claims.**

Technical debt will be avoided with the use of automated testing. The automated testing will decrease the time and man power necessary to test software code which will also decrease the time a project will take to get to the customer.