637 - Dependability and Reliability of Software Systems

Lab. Report #2 – Requirements-Based Test Generation

Group Number#	29
Member 1	Armin Zirak
Member 2	Sepehr Pourabolfath Hashtroudi
Member 3	Mingruil Yu

1 INTRODUCTION

In this work, we test the following two classes of the project.
- Range.class
- DataUtilities.class.

We use the following libraries for testing
- JUnit for Java testing
- JMock for mocking dependencies

since the DataUtitilties is also dependant on other classes that are Values2D and KeyedValues, we use JMock as a tool to mock them so that their fault do not cause irrelevant failures in testing DataUtilities.

2 DETAILED DESCRIPTION OF UNIT TEST STRATEGY

We divided input ranges into partitions based on the specifications. Then, we implemented ECT test strategy to cover partitions. In addition, we added boundary, robustness and worst case testing to localize more faults. Since, the full coverage may go to infinite test cases, we write them as much as possible.

Please note the full test plan strategy is located here (in testPlan file in github repo)

Benefits of using mocks

1. **Bug Localization:** We know that if a test is failed it is because of the class under test not its dependencies. So, we can accurately find the responsible function and solve the issue.
2. **Cost of Calling External Methods:** sometimes, calling dependant functions is not free. For example, they are changing a database, or calling a remote API that has some charges or is limited. By mocking, we can simply solve these issues.
3. **Unexpectable Return Values:** sometimes methods of classes, return the values that are not proper for our testing. Or it may be hard to cause them to return the exact value that we want. Mocking help us solve this issue simply.

Drawbacks of using mocks

4. **Untested Interactions:** the main drawback is that we cannot test the interaction between functions. If we are calling a function from a mocked method in a wrong way, our tests don't cover them. So, we need an additional vs integration testing.
5. **Development Efforts:** Moocking adds lots of additional effort in the work. You should define the return value of each function that increases development time significantly.

3 TEST CASES DEVELOPED

The full description of test cases are explained in the test plan (here)

Test cases are designed based on 'test-case degine' lecture. We first defined the domain of input variables by dividing to expected and unexpected partitions. Then, we further divided each of them into subpartitions. Then, we added ECT (equivalence class tests), and then based on the resources we added Boundary, robustness and WCT tests.

4 HOW THE TEAM WORK EFFORT WAS DIVIDED AND MANAGED

To start with, Mingrui took the responsibility of Range.class, Armin and Sepehr took the responsibility of DataUtilities.class.

We started by reading strategies, preparing test plans and then implementing them. Finally, Armin and Sepehr convert test plans to MarkDown documents (here).

To improve the code and learn more, we review the test plans and tests of each other before finalizing.

We used Github for remote collaboration. Each of us has one branch and gradually merge codes into the master after reviewing them.

Finally, we added reviewed test plans and report to the github repository.

5. DIFFICULTIES ENCOUNTERED, CHALLENGES OVERCOME, AND LESSONS LEARNED

Mostly, the difficulty was to learn mocking syntax for JMock because it is weird and much different from other libraries/frameworks. The community is limited making it hard to find solutions for bugs. We overcame this difficulty by looking at code examples.

Second, the remote collaboration is hard when we want to merge codes. Especially, in this case because the codes are in the same files. So, we used github to fix the issue by having separate branches.

Finally, Eclipse is a bit outdated. Although Mingrui could use it, armin and sepehr couldn’t install it on ubuntu and macOS systems. To solve that, we changed to IntelliJ. This caused additional problems, because the structure differs with the structure of Eclipse on Mingruis laptop. Then, we were to manually merge codes to solve the issue.

6.COMMENTS FEEDBACK ON THE LAB ITSELF

The project is a good tutorial on Java, JUnit and JMock. It teaches the concept of testing and mocking and introduces the challenges of making them practical. However, the workload is far more than the expectation that almost took 1.5 day from each of us.

Thank you!