**Grand Strand Systems: Unit Test Reflection**

Lucas A. de los Santos

Southern New Hampshire University

Software Test Automation & QA

Professor Joesph Rangitsch

June 25th, 2022

**Summary**

The customer, Grand Strand Systems, has tasked us with developing a mobile application. The customer provided requirements for the various classes within the mobile application. Based on the given requirements unit tests were written in JUnit 5 to verify that the code meets the given requirements. The Contact, Task, and Appointment classes all adhered to the requirements by various methods. All classes required a unique, unmodifiable identification string that was no more than ten characters in length. To guarantee automated and unique strings the RandomStringUtils package was used to generate random strings using only letters and numbers and a set length of ten.



Within Figure 1, the Java setters in the Contact, Task, and Appointment classes I’ve made conditions such as cannot equal null, setting length requirements, and including error messages should someone attempt to do something that’s outside the scope of the requirements.

Test coverage was run and checked via the Maven plugin Jacoco which generates a readable code coverage report as part of the Maven package liftcycle. By default, it outputs an HTML file and gives a detailed breakdown of areas that are covered and what lines or branches are missing. In terms of code coverage, the line coverage report the score was 100%. The report from Jacoco details each element within each class and verifies that a JUnit test covers each method. The JUnit tests created check for null values, length requirements, usage requirements, and lastly the add, delete, and update actions.



Figure 2 shows an example of a JUnit test that creates a null taskId and refers to the setter shown in Figure 3. The test sets the taskId parameter to null to invoke a failure based on the conditions in the setter.. When the if condition of taskId equals null it expects a thrown IllegalArgumentException. The test expects the argument output “Task ID cannot be null” and would succeed silently without a System.out.println.



Utilizing Maven’s surefire plugin and the verbosity written in test, the output within the run displays the number of tests including the printed feedback from the test. An example of this output can be seen in Figure 4.



**Reflection**

**Testing Techniques**

The software testing techniques utilized in this project consisted of methods from dynamic testing. The major methods are unit testing and code coverage. Code coverage and unit tests go hand in hand since code coverage percentages are determined based on the depth of unit tests written. Unit testing takes small chunks of code and tests their functionality such as testing a single method. In terms of static analysis, I could have employed more such as checkstyles, pmd, and spotbugs within the maven lifecycle to get a better coverage basis. Due to working on this alone a lot of the other static testing options were not available to me such as various forms of peer reviews. As for dynamic, as the code progress then other methods such as Integration, System, and Acceptance testing can be added. However, since no main class exists and the program itself isn’t runnable beyond unit tests none of these options are available as of yet.

Static testing from a practical level allows the developer to have other eyes on their code and ultimately get a review. Since in-depth knowledge of various aspects of different coding languages is widespread the intake from others can prove valuable. As an example, a reviewer may suggest a better way to write a method or notice a potential security risk within what’s written. Static analysis, such as checkstyle, linters, etc, are valuable assets to help developers keep to certain standards when writing out code. Checkstyle, for example, utilizes a set of rules to refer to when scanning code and then prints out a report with all violations encountered. In terms of dynamic testing, unit testing and code coverage help developers catch bugs before they start. Including a full suite of tests against all their code, it ensures the methods work as they expect and should catch any unforeseen use cases. Code coverage is a determination of those unit tests and gives the developer feedback on how much has actually been covered by the unit tests.

**Mindset**

Due to working as a Release Engineer and a Sysadmin in previous roles, I’m very accustomed to constant “well it works on my machine” from developers in the building. Though low and behold they custom added some dependency or library that wasn’t directly imported into their code or accounted for in any way. With this in mind, I do my best to cover all bases such as clearing my Maven repository cache and making sure things build from a clean state. I utilized the Enforcer plugin to make sure the proper version of Maven is used and set the dependencies to specific versions so as to not get floating version numbers. I have to keep in mind though that I can try my best however without feedback from other individuals and testing outside myself I will not get a fully unbiased review. From a test case standpoint, I attempted to be as granular as possible but even after completion, I saw places where I could have gone deeper into potential test cases but this would have also required more methods in the classes.

Being disciplined is an extremely difficult task depending on the environment you work in but it’s always something to strive for in all the work you do. Being as complete with writing unit tests, planning out your code, adding various tools to help aid you in static and dynamic testing, and a plethora of other boxes to be checked. One of the main difficulties with this is really about the culture where you work. A lot of places prefer things to be quick and dirty as it may be an urgent matter that doesn’t always allow for the slow and right way. Do I agree with this approach? No, and have seen it bite people in the ass so many times it’s not even funny. Therefore in all projects, I take on I set the expectation that it’ll be done when it’s done since I’m taking my time to ensure that it’s done to an exceptional level of completeness.