**CS-320** **Project 2 :   Grand Strand Systems Testing**

Shane Flaten

CS-320-X6369 Software Test Automation& QA 21EW6

Aug 7 2021

Instructor**:** Joseph Rangitsch

**Summary**

**Unit testing and software requirements.**

The software requirements were as follows: The 3 classes of contact, appointment and task required a unique ID first name last name, address, description and phone number that can’t be longer than, or shorter than a certain number of characters all of which could not be null. There was also a date that could not be before the current date. All these elements had to be able to be added deleted or updated per contact ID. For each software requirement there was a special Junit test in place to test each requirement. I used the software requirements to write each Junit test case for each single use case. For instance, I used the standard Junit test to verify that an exception is thrown for each condition statement made with “if” statements. Like in the case of a string being too long; If the string was too long in the object being passed to it, the exception was thrown, and the test verified each logic test made. In other words, requirements are the features software has. This was used as a template throughout the build. To achieve this elicitation was needed after each stage of the build. When a new software requirement was presented, an analysis took place then a design, then implementing, and finally testing it. This is the agile process of delivering after each story which is derived from the requirements. The requirements came from elicitation from the client. My approach was to take the agile process which is part of the software to development lifecycle.

**Effectiveness in quality on the JUnit test.**

After I ran all test, I saw that I had at least 80% coverage on all the code I ran. This is a technique called white box testing. For the “date()” I wanted to verify that I had set the test up right so I made an additional test to verify that the exception was thrown with a date that was set before today’s date; by means of passing the object to the logic test to verify that an exception was being thrown. I use standard test patterns that are known to work. I also static tested the code before I ran dynamic tests to see the code and check if there are errors. I ran the tests as I was programming to see if the code was working before pyramiding up to more complex tests, this kept technical debt to a minimum.

**The experience of writing the JUnit test.**

**Ensuring the code was technically sound.**

I had good knowledge of the software requirements to help guide me in the programming. I also used standards and good practices for object-oriented programming. This included using single purpose classes, like the constructor where the class creates the object, setters and getter that are accessed in a limited fashion to keep from code modifying all over. The methods were easy to read without too many arguments, and a reduced reliance on global. I did not move on to another piece of code until I had done an incremental test procedure.

**Efficiency of the code.**

The code was efficient is using setters and getter. I also used the @BeforeAll to make the object for the test rather that making the object each time I ran a single test case. I also used a reusable code to: retrieve the length, and to loop through the list to find the Id rather than writing it every time. I also used a generic object, this object when called brought the last object called, this way the test did not need to get the exact object ID just the last one created. This generic is also common and well used in the industry.

**Reflection**

**Testing techniques.**

**The characteristics of the testing techniques used.**

I used the JUnit incremental test to increment testing as I coded. Unit testing is when each model is tested one at a time requiring access to the code. I also did integration testing to look for defects in the interactions for instance, when accessing the list with the test and modifying an object. This was a type of regression testing that ends for each sprint. I used the Agile method of releasing completed and tested code that was released on a timely basis, rather than the waterfall method of all testing done at the end. I used the technique of testing often and testing early. After the debugging of errors, I used systematic testing, an exploration of each component in the system that defines defects. First, I planned the test to see what areas I could control; this involved mapping the components and determine what targets I needed to meet. l then I analyzed to see what test conditions were being tested and what should happen under certain circumstances. For instance, an object first had to be made before the other test could be performed. The testing code was made with known patterns and designs, then I combined all the test cases into a run procedure. Then I implemented the test and evaluated the exit criteria finally reporting. While static testing I looked for defects in the software without running the code I did this frequently looking for things like unused variables dead code and loops that don’t work. This is done because you can’t find everything with dynamic testing if something just doesn’t run. During Blackbox testing I used equivalence partitioning based off the requirements that were gained from elicitation for the application. This meant taking the input of the domain which was the string length of the input. This insured that any integer greater or less than a certain number would not pass, and an exception would be thrown this could be seen in the JUnit tests for each module. Anything that was tested could be put into two categories invalid or valid. I tested both valid and in valid equivalent classes. One of the requirements was that it could not be null, so I made a test just for that. I also used white box testing which is the test coverage and had over a percent requirement in all tests. In some cases, I had 100% test coverage but never below 80%. The unit tests will test the internal structure, it’s a type of regression testing for each module so you don’t have technical debt. This dynamic testing of the JUnit test allows you to take an integrative technique and more agile approach. I did use a decision table that was very simple just to keep track of the JUnit tests that I was writing.

**Testing techniques not used.**

I did not use the water fall method to reduce technical debt. There’s also the spiral model the model of the big bang model, in the rapid application to deployment and prototyping models.

While static I did an informal review as opposed to a formal review because this project was small and simple and I was the only one coding there’s no reason for a formal review, if I had had a lot of external libraries used that might’ve been more important to determine vulnerabilities. I didn’t use anything other than agile development because I was just meeting the requirements at a bare minimum to meet the user stories. I wanted to increment the deployment of working code on time. If I were to use devOps dynamic testing I would’ve had more difficulty because I was not comparing expected input and output values as they were already built into the test this would’ve made the process much longer due to constant collaboration between the programmer and the tester. This was unneeded because it was just me being responsible for the code. I did not need to do too much integration testing other than just seeing if the service modified the class object, and I never called external classes that needed to integrate or change the state of the system. I didn’t do any system testing because there was no Main. I did not do acceptance testing because there is no end-user beta testing in a Q&A team. As far as non-functional testing I never tested the performance / load under various conditions because there were no stability issues with a simple object. There was no security in this project so there was no need for that, but I did use Secure coding practices which were part of the review and planning process. I did not have usability testing because there is no user interface to except the input after a certain stage. I did not use compatibility testing because there were no systems or platforms that the functions had to support. No need for state transition testing because application never change states under a certain test. Although an object had to be created first it was relatively simple and only the array changed in size. I did check the size of the array to see if the object was put into the array. As far as Personal user case testing, there was no main class or a user interface.

**Practical uses**

I could’ve used boundary analysis which is just the exact boundary at which something does not pass, but I did not use that black box functional technique because I had null as an invalid input. Another reason is because with the date method I also used a time method so you could make an appointment that same date but not the same time, it would’ve been impossible to get the exact second for a boundary analysis. For static testing I did a once over a review of the acceptance criteria and did not need to do it after the first story. The reason I did not do a formal review is the team was only one person and it was relatively simple. I could not use the waterfall method because there were some changing requirements later. I thought the waterfall method could’ve came technical debt in larger projects, but it would’ve been feasible in this project due to the repetitive nature of the classes. DevOps would’ve been another choice that I did not use because again the repetition of the classes and the simplicity of the code meant I did not need that much security for complex attributes. This would’ve been a better choice if the code had many different programmers with many different security requirements but after the first iteration of the user’s story there was no need for collaboration. If there had been a main and user interface there would’ve been another layer of testing that would’ve been needed.

**Mindset**

**Mindset adopted, caution, and interrelationships.**

I used a mindset of focusing on the individual attributes of the code, testing each functional aspect of code and focusing on just that. I was cautious when using naming conventions, this kept my code easy to read and I could easily remember what function of method to call to use in the function I was working on. There was a step-by-step process that I need to employ because the object had to be built right in order for the methods to work. I then used the software requirements to make each modifier from the tested patterns. I used best practices for object-oriented programming, for instance in my code I use setters and getters to limit the access to the object. I also use a constructor too initialize the object. These best practices and patterns are key to being able to walk away from the code and know that it works because it has been tried and tested by many programmers. Or when making the service classes I made sure that the class of service was modifying the object as I programmed and tested the code. In a sense I was using integration testing to see how the component of the object reacted to the object’s modifiers. It was also important to use patterns to do the complexity and interrelationship of the code. For the first class I made in this application I made sure to over comment the code so the following classes that use the same pattern, I could read the comments on and know exactly what is going on in the code. This help me be careful methodical in writing the classes. Finally, I removed the unnecessary comments so that the code was easier to read for future programmer, debugging, and testing. These comments and careful naming conventions helped isolate code if the function what’s not working while performing dynamic testing. This mindset of a step-by-step process to be thought of as an iterative process that goes through the software development lifecycle in a methodical timely manner. This is necessary to deliver working code and understanding the relationship among modules in the code.

I tried to limit bias by using a standard or patterns of code. This way I could check to see if the result was processed properly. This way I did not need to worry to much if the code was technically sound, because patterns are a way of using something that is tried and tested. But this could also lead the bias, I reuse the same pattern after I found the first pattern worked, but I had to make myself responsible for the code and run a test on the following classes. If I had not done this, I would’ve missed a name in error. To avoid bias, I use the standard software development life cycle framework which is an international standard were testing activities are involved in all stages. Another thing I had to do to limit biased; working on the time module in Java for the date, after was completed I ran a test and found it worked, and then realized that the test itself may have a flaw. I built another time object and ran a test just with that object, this confirm that my test was working. In other words, I had to use more than one test in technique, if I had been biased towards a certain testing pattern, I may have missed a flaw in my code. Because I’m responsible for the delivered code I’m also responsible for the tests, it’s very important for the entire team to be responsible for the code and the test that’s why DevOps is needed in some cases.

**Discipline and quality.**

It is important to not cut corners while writing code because if you cut corners the code becomes broken and hard to de-bug. It is important to stay within the standards of naming principles, and have correct comments on the code, because if the code is not readable it is very difficult to work on in the future. Naming standards help track what methods are being called from certain classes and what they do. This way it’s easy to increment the debugging process, allows an increase of flow while writing the code. Another very important aspect is testing, it’s a methodical process of testing is implemented using the standard procedures, it ensures a working product that has been tested under live conditions. This is important for the client because a simple mistake could cost the company and the livelihoods of the people working there. This could also impact the consumer negatively in such a way that defected parties may not be able recover. Example would be the explosion of the unmanned Ariana five rocket that was launched; a floating-point number was converted to an integer that ended up being too large to be stored, so the conversion failed causing an accident. This is a simple mistake that could’ve been avoided if testing had been done. But more than just boundary testing would’ve had to been done so not cutting corners on testing is very important in some cases. This could lead to bugs and vulnerabilities in the application. So, maintaining the standard workflow procedures for testing is an important discipline to maintain, even if it seems trivial it may not be in the future. Event while static testing doing a formal review for a critical component of the application may be time consuming but necessary. This takes a discipline mindset. Doing this early on and often reduces technical debt. In the future while working with a team I will use a devOps to avoid going too far into the code or programming a component without using testing procedures. DevOps is a great way to collaborate with the testing team and makes everyone responsible, this is insurance that the product will be delivered without errors. For instance, while working with the job, a time object it was important to read the official documents, plan based on specific known patterns that work, then implement the code, finally test to see if the output right. In this case I had to even test the test I was using to see if the test itself was doing what I wanted to do. After confirming everything I had working code.

References

Brian Hambling (Editor), Peter Morgan, Angelina Samaroo, Geoff Thompson, and Peter Williams. (2015) *Software Testing: An ISTQB-BCS Certified Tester Foundation Guide 3rd edition.* BCS The Chartered Institute for IT

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

McGlaufin B. (jun.25 2019*) Unit Testing Best Practices: How to Get the Most Out of Your Test Automation*

https://dzone.com/articles/unit-testing-best-practices-how-to-get-the-most-ou

*Best Practices of Object-Oriented Programming (OOP)*. (June 2020) https://www.geeksforgeeks.org/best-practices-of-object-oriented-programming-oop/.

*The Explosion of the Ariane 5*. https://www-users.cse.umn.edu/~arnold/disasters/ariane.html

Brian Hambling (Editor), Peter Morgan, Angelina Samaroo, Geoff Thompson, and Peter Williams. (2015) *Software Testing: An ISTQB-BCS Certified Tester Foundation Guide 3rd edition.* BCS The Chartered Institute for IT

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