James Jones

Kalysa Wilson

CS-320

8/13/23

Project 2

In order to align my testing approach with the specific software requirements I had to make sure all the requirements get atleast some coverage.

IllegalArgumentException thrown1 = Assertions.*assertThrows*(IllegalArgumentException.class, () -> {ContactService.*addContact*("123456789789456", "John", "Smith", "1112223333", "1234 Main ST");});

*assertTrue*(thrown1.getMessage().contains("Invalid ID"));

IllegalArgumentException thrown1 = Assertions.*assertThrows*(IllegalArgumentException.class, () -> {ContactService.*addContact*(null, "John", "Smith", "1112223333", "1234 Main ST");});

*assertTrue*(thrown1.getMessage().contains("Invalid ID"));

Here are two separate lines of testing the Contact class for invalid ID’s this includes too long or null ID’s. This was implemented for each of the variables present in the system ensuring that all variables comply to the length and null requirements set out for the system. By testing every variable this way instead of just one or two we are able to ensure that no mistakes were made between variables, maximizing the overall test coverage of the system.

Writing technically sound code is important so before any code was written a large amount of research was done. The provided YouTube video about Junit testing was revisited numerous times to make sure I was on the right track. After this checking that no errors were being caught by the IDE also helped me clean up my code.

ContactService.*addContact*("1234", "John", "Smith", "1112223333", "1234 Main ST");

ContactService.*addContact*("1235", "Jason", "doe", "1112423333", "1235 Main ST");

ContactService.*addContact*("1256", "James", "Jones", "1112223533", "1236 Main ST");

*assertTrue*(ContactService.*contactList*.size() == 3);

ContactService.*deleteContact*("1234");

*assertFalse*(ContactService.*contactList*.size() == 3);

In this code snippet I added 3 values to my contact list then tested that there were indeed 3 values in the list. This tells me whether my addContact function and my arraylist was operating correctly it also gives me a known value to go into my deleteContact testing. Next I delete a contact then again check my array list this time making sure that there is not 3 in the arraylist. Basically I gave myself a known value to test against so that I could then run my test and validate the answer manually allowing me to double verify my code. Ensuring my code was efficient came down to checking my test runtimes making sure that nothing was running excessively slowly.

I used numerous different testing techniques such as manual inspection, unit testing, and functional testing. Manual inspection is looking through code line by line to check for issues. I found numerous issues during the process of writing my code that manual inspection was able to turn up. Functional testing was simply ensuring that I captured all the requirement given to me. Finally was unit testing which was my Junit tests where I tested individual functions of the code to make sure that it was working correctly.

There are many different testing techniques that I did not use for this project including non-functional, integration, system, and acceptance testing (Hambling et al., 2019). Non-functional testing is testing tests non functional requirements like security and speed (GeeksforGeeks, 2023). Integration testing makes sure that the individual units and can be properly implemented together. System testing tests the whole system and then acceptance testing is done with the end user to ensure that the system truly meets their needs (GeeksforGeeks, 2023). In this project these forms of testing weren’t necessary at this stage of the system.

Testing techniques are implemented together in order to get a larger coverage of the system. Functional and non-functional requirements testing when taken together covers the full system requirements of the system. Manual testing allows for checking the lines of code and finding logical issues or missing requirements. Unit testing is great for checking that individual components properly work however doesn’t capture the larger interactions of the system. This is where integration testing comes into play by testing that the components properly integrate, this doesn’t necessarily check requirements. System testing then checks the overall system making sure everything is working properly and that the requirements laid out have been captured however doesn’t include the end user leaving a gap in the testing process. This is where acceptance testing comes in here the end user is involved ensuring that all their requirements are captured and that the system operates “in the wild”. Together these tests capture most of the system theoretically full coverage. However, like in this project many of the different testing techniques weren’t suited for the stage of the system that I am at but would later on be incorporated if the system were to go on further.

The mindset I adopted was a slow and methodical one I wanted to understand exactly what I was doing and why it was necessary. On the first milestone I was attempting to use some coding techniques that I had never used and got lost quickly in the code. This isn’t the most complex code but if you don’t pay attention to what you are doing it can become very confusing rather quickly. Especially as the system becomes more and more interrelated the way a system communicates is incredibly important in order to understand the system but can be very easy to lost track of. The way I try to eliminate bias is by basically assuming that everything that I coded was wrong. This means I can’t skip a line or assume something works because I know it’s wrong until I have proven that the specific component is working. I adopted this strategy after the first milestone where I went in confidently and had a bad time the code I wrote didn’t function properly. I changed strategies for the next milestone and had far more success, bias makes it very difficult to retain this type of thinking especially as we moved on to the third milestone and the project. By that point I had turned in other assignments all done correctly it would have been easy to start cutting corners to get things over with so I could do other things. This is where bias can be such an issue anything can feed into that mentality. This is where discipline comes into play during the discussion this week looking at released escapements there are major real world consequences to these bugs or defects. Ensuring that corners aren’t cut and that you have truly done your best is incredibly important in ensuring that your code is quality, this is also the ethical decision.

Bibliography

Bibliography

GeeksforGeeks. (2023, February 6). *Software testing techniques*. GeeksforGeeks. https://www.geeksforgeeks.org/software-testing-techniques/

Hambling, B., Hambling, B., Morgan, P., Samaroo, A., Thompson, G., & Williams, P. (2019). *Software testing an ISTQB-BCS certified tester foundation guide - 4th edition*. BCS, The Chartered Institute for IT.