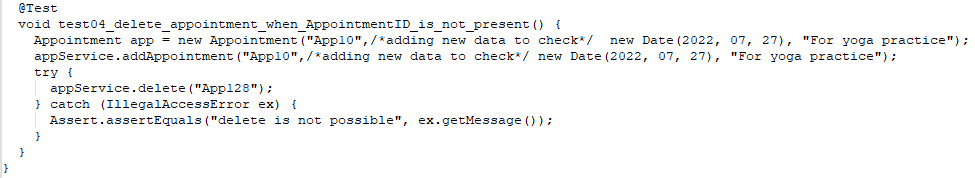
# CS 320 Software Test Automation and QA

***7-2 Project Two Submission: Summary and Reflections Report.***

**Summary**

Each of the three features (Contact, Task, and Appointment), had specific requirements all of which required as stated in my previous reflection “testing for runtime errors based on the (dynamic input) requirements (i.e. the ability to add, update, and make deletion in the Contact/Task/Appointment objects within the application). The requirements were focused on functionality than internal implementation (efficiency,security etc).

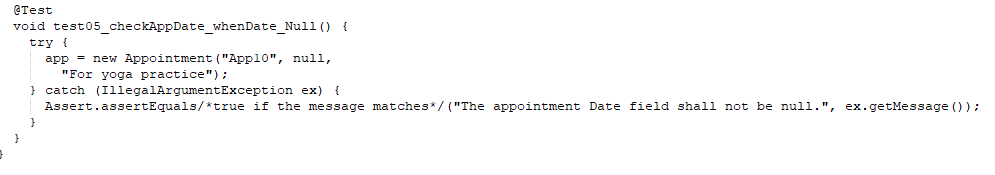
My unit testing approach was therefore aimed at validating that the individual classes (add, delete and update) functions ran independently as expected based (for example) on the Contact object (firstName/lastName characters, phone string digits, address Fields) as well as the Contact service (firstName,lastName, Number, Address update) requirements. For the add, delete and update templates, I created 3 tests using known variables such as with unique contact details/values and then made changes, additions, and deletions in my unit tests true, false test method to rely on or test based on those defined parameters. An integrated test was also ran for when there is a parent and sub classes, as well as a regression test to determine the outcome for when there was a change in configuration.



For each class the aim was to have a coverage of 80% of each implementation method /class on each of the requirements. I was able to attain 80% and above hence proving my code quality and stability were sustainable. This also meant my tests were covering test requirements, code, as well as the user (add, delete, update) scenarios for the app.

In terms of technical soundness of my code, I attained this by reviewing the functions as per the tests to determine the root cause and source of the errors. All (+/-) results had to reflect the requirements specifications which asked for the ability to add contacts with a unique ID, delete contacts per contact ID, update contact fields (first/last name, number, and address) per contact ID. There were also requirements or restrictions on the Contact class objects (first/last name string field length, phone string field, and address field) that had to be met. In order to ensure that my testing approach aligned with the above requirements, I made sure that my test class included a test method contained in the class that had enough coverage. For example for Contact Service, I made sure to test the Add methods , Delete methods, as well as test the Update Methods, all of which included adding/deleting/updating then testing to ensure those changes threw the desired (error) message.

My code was written such that testing for errors would be based on the (dynamic input) requirements (i.e. the ability to add, update, and delete Contact/Task/Appointment objects within the application). So, code efficiency in my opinion included bearing in mind that unit testing will be centered around validating that the add, delete and update functions ran as expected based for example on the Contact object (firstName/lastName characters, phone string digits, address Fields) as well as the Contact service (firstName,lastName, Number, Address update) requirements. For the add,delete and update templates, I made sure to create at least three tests with unique contact details/values and then made changes, additions, and deletions in my unit tests true, false test method to rely on or test based on those defined parameters.



**Reflection**

The app as per the specifications, requirements and functionality required a structural testing technique with JUnit in the code. The requirements made it appears as though Black Box (dynamic) testing would be most appropriate, especially as there were no dependencies in regards to internal implementation. The process was to include a Junit library in the project, then create a new Junit test case in the Java IDE, and then running the test case. The object was that the test would return a true result if the objects are equal, false if the passed object is null, or if the getClass() method is not equal.

While I made sure my JUnit testing were defects free as per the requirements and functionality (in White Box Testing), I did also extend testing for Boundary Value Analysis, Equivalence Partitioning etc. This project involved automated testing technique since we used code and scripts to automatically check the app. I also implemented some exploratory testing to make sure my code met standards and was functional. Having clear requirements also meant using checklist-testing technique to ensure we were in alignment (from a quality and requirement perspective) with the client’s needs.

As I mentioned in my previous write up, there are several other useful software-testing techniques that simply would not have been as helpful considering the current stage/state of the application (i.e. as of Module 6), as well as the customer’s requirements. Under Unit testing for example, the Gorilla Testing would only be helpful if all the “modules” of the application were already in place. I also did not see the need for Integration Testing since we are not testing modules of the application that are being grouped together at least not yet. I also did not use System testing since we only focused on features/modules and not the entire application or system as one whole functional unit. Acceptance Testing (UAT) obviously would not happen until later when the application is somewhat complete (with a UI etc.) hence no need to focus on that at this point in software development process. Other testing techniques include:

* Gray Box is a testing method which is a combination of both Black Box and White Box Testing and considers a scenario where tester tests the product as someone who either fully or partially understands the implementation logic or not at all.
* Ad Hoc Testing, is used to test a product without any planning or documentation serving as a guide. This type of testing is also called Random testing; this method is normally used during Acceptance Testing.
* Agile testing is a technique used during the testing of agile developed software. This technique accommodates the testing of every changing software.
* Dynamic testing; involves testing code while executed. This technique is ideal for testing complex coding.
* Static testing; involves testing code without executing it.
* Unit testing and its different test techniques like the ones mentioned above would be best during the development phase to ensure unit component defects are caught, and addressed at an early stage. This helps in additional cost avoidance, and resource/capacity planning, especially in a waterfall environment where adaptability and change is not quite welcomed.
* System testing will be vital to the overall operability of the application as a whole against the specified requirements. Acceptance testing includes real time business scenarios. This helps in ensuring that the application’s functionalities from an end-user perspective are as per the requirements. This typically happens last and/or before the application is deployed or before go-live. There are also non-functional testing techniques such as security related (penetration) testing that are also helpful in certain projects.

**Mindset**

My mindset was one of caution and care through all the phases of this project because I wanted to be able to deliver code that was logical and functional. Caution therefore had to be exercised to the fullest extent and/or at just about every phase of the (dev/testing) process, because not only did I have to ensure the I fully understood each functionality, each also had to meet the requirements as defined (by the customer) and/or from an end user perspective. The complexity and Interrelationship of the code gave me the ability to put my testing/coding ability to the test to ensure that my code will be functional upon runtime.

By ignoring the fact that I was testing my own code and focusing on making sure that it was truly functional and met the requirements as per the customer was I think a big take away for me in terms of bias.

As a programmer, the importance of quality can never be overemphasized because code in many cases can be very impactful in people’s lives, it can ruin the reputation of an organization, cost huge financial loss and tarnish your image as well. Remembering that one is serving a client and the quality of ones work represents more than just that individual’s reputation. Quality can also be source for good motivation and an inspiration to others especially aspiring programmers.

Delivering a good quality product that meets the customers’ requirements will always be the best way to avoid and/or eliminate technical debt. As a programmer the quality of service, the ability to own mistakes, fix them in a timely fashion; communicate discrepancies, dependencies and/or subpar work will all help towards avoiding technical debts. Additionally being able to not assume that my work is correct or ready without peer review, proper testing also will be will all help in give me able to provide quality work and maintain a good reputation. Overall, writing code comes with a certain amount of responsibility which involves being able to deliver nothing less than the customers’ expectations.

**References:**

Gulati N, (2021, March 1). *Software Testing Techniques*. GeeksforGeeks

Retrieved from <https://www.geeksforgeeks.org/software-testing-techniques/>

Kind regards,

Denis.