Project Two

Shelby St. Clair

CS 320

I made sure to keep the project requirements in mind every time I added a new function into my code. If I chose to ignore project requirements or build my project without the requirements in mind, the end result of my project wouldn’t meet customer or client needs. For the ‘ContactTest’ class, I used parameterized tests for all of my JUnit testing requirements – using the parameterized tests allowed me to test for all values (including null) without needing to build a separate function for each one. For example, to test the Contact ID, I built a parameterized test using “1234567890123” as the ValueSource (as the length needed to exceed 10 characters) – then, I added a NullSource to make sure it wouldn’t accept a null value for the Contact ID. Then, I added a new contact using the contactID call rather than inputting a hard value – this runs the test using the long ValueSource string and the NullSource to make sure neither can be assigned to the Contact ID. I used the parameterized tests for all JUnit tests that allowed for the parameterized test, which helped contribute to the overall 85.5% coverage of my application. One of the most difficult parts of coding this project to the specified requirements was properly using the LocalDate call in the Appointment class. I found that using the “LocalDate.now.plusDays(1)” call made it easy to make sure an appointment is future dated – for the JUnit tests, I used “LocalDate.now.minusDays(1)” to check for any date in the past. Using these calls should ensure that the program will run regardless of what date is used – using a hard date (ex: 12/31/2026) would cover a large period of time, but would need to be updated once 12/31/2026 occurred or else the program would no longer function properly. As I mentioned earlier, my total coverage percentage for my JUnit tests is 85.5% - the high coverage level lets me know that my JUnit tests are successful overall. While certain classes in my project have a lower coverage percentage compared to others (for example, my AppointmentTest class has a coverage percentage of 63.4%, while the AppointmentServiceTest has 100%), the overall coverage shows that most of my tests have met project requirements effectively. The JUnit tests I created allowed me to quickly and efficiently test my code using a multitude of different inputs without needing to manually build and input test data – it’s a very helpful method to ensure the code is sound and meets all the necessary requirements for the project to be successful.

I made use of several different testing techniques in my project – my unit tests were all automated functional unit tests, meaning each individual function of my code used was tested using automation (in this case, JUnit testing). Integration and Regression testing was used, as well – each of my test classes had a corresponding class holding the parameters and functions that needed to be tested, and I implemented changes and adjustments based on feedback from assignments I submitted and tips I read about online. The project was ran each time I made a change to make sure it wouldn’t have any negative affect on the rest of the project’s functions. There were several testing techniques I didn’t use, mostly due to the nature of the project. I didn’t use any System testing, as there wasn’t a whole system used in the project – therefore, System testing was unnecessary. Performance testing was employed very minimally, mostly to ensure the code is sound enough to avoid a large number of changes down the line. I also didn’t use Security or Exploratory testing – this stage of the project didn’t have any type of security requirements, so there was no need to complete Security testing. Exploratory testing involves running tests without any type of formulated plan, which I did not do with my project – all of my testing and code was thoroughly planned out before beginning.

Other projects that have more in-depth testing requirements would make great use out of most, if not all, of the testing requirements I mentioned. Comparing two different types of projects (an online video game and a database that holds government employee information) can show how these testing techniques can make an impact. Both of these projects would utilize the testing techniques in different ways - for example, integration could be used to test the player’s settings between the menu options and actual gameplay, and to make sure an employee’s access is set up properly based on their job code. Regression testing can be used to make sure nothing else in the game is broken if a bug fix is implemented, and to make sure no systems go down for the government employees if a system update is initiated. System testing would be used to make sure the game functions on multiple console types, and to ensure changes made to employee data in other connected systems reflect as needed in the database. Exploratory testing wouldn’t be as necessary in the government database, but the video game could benefit from exploratory testing by doing random inputs/exploring random areas in the game to see if any fixes are needed. Performance testing would be necessary for both - the video game needs to ensure the players experience seamless gameplay with no lag to keep the game fair and enjoyable, and the database would utilize performance testing to make sure changes to employee data are updated quickly and accurately. Security testing would be necessary for both projects, as well - player profile information needs to be kept secure, especially if payment information is linked to their profiles. The government database would need to ensure unauthorized users aren’t able to make changes to employee data, and that access is adjusted/revoked if employees change positions or leave their position. All of these testing techniques can have other applications for each of these products as well, and all will help make sure the product meets the requirements and is both secure and efficient.

I employed a relatively high level of caution when building my final project. Once I made sure the first section (Contact) had a good coverage level, I wanted to make sure I had uniformity between each section of the project to maintain the same high coverage. I also made sure that my code functioned as necessary, especially when building the LocalDate functions in the Appointment classes and when creating the Hash Map in each section. With the LocalDate functions, I ran into trouble implementing the parameterized tests - rather than risk the program not functioning, I chose to implement the tests in a different format to make sure I stayed within the program requirements. I don’t work with Hash Maps often, so I took great care when building those into my project - I had to go slowly to make sure my functions matched the output and that I used the key functionality rather than trying to sift through a list. I tried to limit bias as much as possible in my code by focusing on the project requirements as much as possible, and keeping the requirements in mind whenever I added new lines of code or made any changes. It can be easy to stay within your own personal comfort level when coding, especially if you think it’s easier. I tried to make sure I researched all the best ways to implement the requirements, and implemented all of the feedback I was given in previous module submissions to strengthen my code. I can definitely see how bias can have a negative effect when testing your own code - there could be multiple situations you don’t consider that aren’t covered during testing. I think keeping an open mind, having adaptability, and putting the requirements first can help keep bias from becoming an issue that negatively affects the project.

Maintaining discipline when it comes to quality within code is extremely important. Meeting the requirements isn’t necessarily enough - it’s equally important to make sure the code is secure, efficient, and built to last (meaning the core aspects of the code can easily be maintained if the product is expanded or updated down the line). Cutting corners can help with meeting deadlines, but meeting deadlines won’t matter if the code isn’t sufficient or effective. It’s better to ensure quality code is built and tested as thoroughly as possible before moving on to future steps in the SDLC. Incomplete or inconsistent testing can result in a large number of system-breaking bugs and defects that can cost a lot of money and time to fix, impeding progress on the project. If testing is done early on and as thoroughly as possible, the bugs can be found and fixed early on - this will help ensure we maintain a high level of quality code that meets all necessary requirements, which will keep the client and customers happy.

References:

Software testing : An istqb-bcs certified tester foundation guide - 4th edition. (2019). BCS Learning & Development Limited.

Boni Garcia. (2017). Mastering Software Testing with JUnit 5 : Comprehensive Guide to Develop High Quality Java Applications. Packt Publishing.