Joseph Iadarola

CS 320

Project 2

The testing phase of the software development life cycle (SDLC) is arguably the most important step because it is during this phase that all requirements for the developed software should be completed. When completing the testing phase, it is important that tests are set up and conducted in a clear and methodical way so that each software requirement is covered during testing. These tests will also need to be viable in the future during the maintenance phase to make sure that there are no bugs with previously completed requirements. There are several software testing techniques such as dynamic and static testing. Static testing is done without running the software code with the purpose of finding bugs and errors early in the SDLC. In some ways, static testing can be seen as a thorough code review process to find bugs and defects. Dynamic testing is performed while running the code to ensure there are no bugs or defects in the software requirements or functionality. This type of testing is generally done during the development phases when parts of the software are completed but during the testing phase, the entirety of the software is tested to make sure that each part works together to an acceptable performance standard.

During the development of the three features, I ensured that I aligned with the software requirements by staying organized. To help with this I used a separate test file for every class and used a consistent naming structure throughout the development of each feature. This naming structure helped to make sure that tests aligned with the proper class. My organization in regard to creating tests meant that as I completed requirements in each class, I would then create a comprehensive test for that requirement. Doing this allowed me to check off a requirement and test in one go without having to go back and check that I did not forget any requirement-test pair. When creating the JUnit tests, I attempted to test that the requirement was complete but also that the given restrictions were also tested such as the limited length of descriptions.

Initially, I thought that I would complete the easy requirements such as creating correct Contacts before moving on to the Contact Service class but then I hit a problem. My first attempt at testing the Contact class did not go well because I set up a test for each requirement without realizing that the tests needed to be combined. I attempted to create a new Contact object for each test when I should have created a single Contact object using the @BeforeEach command and then testing that the object followed the requirements laid out.

Along with the dynamic testing that was used during the JUnit testing classes, I also used code review to make sure that I included the limitation requirements before I started testing. I did not use techniques having to do with documentation because each feature was small enough that I did not feel like I needed to map them out. If there were more requirements for each feature, I may have needed to make a UML diagram or process diagram but for this project, JUnit testing was enough because the requirements were simple and did not require complex processes.

While completing this project, I tried to adapt my mindset to create patterns. I employed caution by creating each class in a linear fashion. Since each feature consisted of an object class and service class, I attempted to have a complete and tested object class done before moving on to the service class. This caution gave me the assurance that when there were failed tests from the service classes that the problem came from those classes and was not related to the object classes that were already done. This did trip me up because I was biased against checking my object classes when I did have problems but that only happened once when there was redundant code that I missed deleting in the Task class. If I was testing other developers’ code, I think I would not have had this same bias because I would not have known how they approached completing the project. Being diligent to the quality of your code I something I think that many new developers don’t have because they have never needed to go through someone else's code and have never had a peer review their work in a real-world situation. In the business world not, everyone has worked on the same project so there needs to be enough tests, documentation and other resources available for a peer or customer to be able to follow what is going on and effectively communicate any issues they have with what has been delivered.