Connor Fortier

Week Seven - Project Two

CS 320

10/17/2021

The unit testing approach used for each of the three features of the clients mobile application, tested the software for and bugs. The approach used to test the software was aligned directly with the software requirements. Specific evidence of this can be found in the source code from the project one submission. Said source code tested each class for the desired project requirements. An example of the requirements tested for the Contact Class include, but are not limited to, a contact objects first and last names being less than 10 characters, a contact objects phone number must be exactly 10 digits, and the contact objects address field may not be longer than 30 characters. Next, an example of the requirements tested for the Task Class include, but are not limited to, a task object ID being less than 10 characters, a task object name that is less than 20 characters, and also a task object description that cannot be null and is less than 50 characters. Lastly, an example of the specific evidence to prove the requirements were tested for in the Appointment Class include, but are not limited to, an appointment ID that is less than 10 characters, an appointment date that is not in the past, and a description that is less than 50 characters. Furthermore, there are more tests that were conducted on each class, however, there is limited time and space to list them all and a general overview was given to briefly describe each claim with supporting evidence. The overall quality of my Junit tests was adequate for the purposes of the assignment, as unit testing is preformed to help find errors early in the software development life cycle (Hamilton, 2021). Out in industry with more time and resources, a much more intricate and secure test could have been developed, however, that would have taken to much time for the purposes of this assignment submission. The Junit tests were effective and showed a 100% coverage percentage, which is successful enough for the assignment.

The experiences I had writing Junit tests this semester was engaging, enjoyable, and I learned a lot. The reason that I learned a lot from writing Junit tests is because the knowledge I gained from the readings was applied hands on during each milestone. This hands on approach to the milestones was where I wrote code for the contact, appointment, and task classes, then after writing that code; I wrote the code for all of the Junit tests which tested each class individually. To ensure the code I wrote was technically sound I did a Few things which were using industry standard best practices, I used proper syntax for the programming language, and also I did exactly what was asked of per the project requirements (this is a very vague question and a more professional term than “technically sounds” probably should be used in future semesters). The following is a citation from the code I wrote for the submission during week six which demonstrates technically sound code in the appointment test class that tests the date object input for a null value:

*//test date is not null*

*@Test*

*@DisplayName("Date cannot be null")*

*void testDateNotNull() {*

*Appointment appointment = new Appointment(null, "Description");*

*assertion(appointment.getAppointmentDate(), "Date was null!");*

*}*

To insure that the code I wrote was efficient, I used industry standard best practices, I wrote variables locally and not globally, and I only tested for what was necessary with no redundant testing. The following is an excerpt of code from the project submission on week six which demonstrates efficient code in the Task Class that is showing the use of industry standard best practices by declaring variables locally and using a constructor for the class:

*//declare and initialize local variables*

*private final String taskId;*

*private String taskName = "";*

*private String taskDescription = "";*

*private static AtomicLong idGenerator = new AtomicLong();*

*//Constructor*

*//public class for task object to validate input for variables*

*public Task(String taskName, String taskDescription) {*

*//Task Id*

*//create id that is less than 10 characters*

*this.taskId = String.valueOf(idGenerator.getAndIncrement());*

*//Task Name*

*//validate input and truncate, if needed*

*if(taskName == null || taskName.isBlank()) {*

*this.taskName = "EMPTY";*

*}*

*else if(taskName.length() > 20) {*

*this.taskName = taskName.substring(0,20);*

*}*

*else {*

*this.taskName = taskName;*

*}*

*//Task Description*

*//validate input and truncate, if needed*

*if(taskDescription == null || taskDescription.isBlank()) {*

*this.taskDescription = "EMPTY";*

*}*

*else if(taskDescription.length() > 50) {*

*this.taskDescription = taskDescription.substring(0,20);*

*}*

*else {*

*this.taskDescription = taskDescription;*

*}*

*}*

There were several software testing techniques that were employed during the development of the mobile software application for the client. Said software testing techniques include, but are not limited to, Integration testing, System testing, and Unit testing. Unit testing was applied to the software application by ensuring each individual component of the application functioned as was intended (Aebersold, 2021). Next, integration testing was preformed to make sure the units functioned with one another(Aebersold, 2021). Also, system testing was done on the clients mobile application in order to verify that the entire system functioned as a whole when ran together as a complete application (Aebersold, 2021). The software testing techniques used were functional software testing techniques; there are also software testing techniques known as non-functional software testing techniques. Some examples of non-functional software testing techniques that exist, but were not used for this project are, performance testing, security testing, and compatibility testing. Performance testing is to test how an application functions or preforms when used, and this test was not done for the clients application. Security testing was not done on the clients mobile application and will have to be done by security professionals before releasing the application to verify that it is secure. Lastly, compatibility testing was not done, and will need to be done, in order to make sure the clients application is compatible with the mobile devices they intent their customers will use the application on. A few practical uses of the testing techniques described above include, but are not limited to, testing individual pieces of software in industry, testing the security of a software application before it is released to the public, and verifying that an application will work, once completed, on all users devices with no bugs. A few implications of the software testing techniques discussed above include, the testing engineers will know how to design a proper test, the software engineers will test the proper input data for said test, and also that the software engineers will document any input that causes errors and then update code accordingly to fix any discovered errors.

The mindset that I employed while working on the clients mobile software application was that I am working out in industry and getting paid to complete a job; even though it was a school project, and I am not receiving any compensation for my time. The extent of caution that I employed as a software tester was a high level of caution. The reason I use so much caution when testing software, is because I was taught to check every program for bad user input because any input that can break a program can do harm to a user’s computer. This is important in industry because the complexity and interrelation ships of the code can sometimes make it easy for errors to be overlooked and when coding professional applications customers have a certain standard of proficiency they expect when paying for a product or service. A specific example of this is if a bank wants a company to program a secure mobile application for their users that does not share password information; however, maybe testing was not done on the application and all of a sudden a bug occurred, and all the users passwords were visible to other users for some unknown reason. A very farfetched idea, but still possible non the less if caution is not employed in software testing. A few ways I tried to limit bias in my code was taking more time than I had to work on testing, testing all cases even if I thought they were redundant, and trying to think like a user who is using the application. Bias could be a concern for a developer testing their own code, who thinks their code is already perfect. I am an amateur programmer and have accepted that the code I write will be full of bugs that need to get worked out. Finally, it is important to have discipline and respect for code when programming any application. This is especially important when programming applications that can impact the safety of lives, for example, the self-destruction of the Ariane 5. The Ariane 5 was a rocket that exploded upon launch due to an error in the programming that was overlooked and ended up costing the company over 7 billion dollars in lost revenue (Arnold, 2000). It is important to not cut corners when writing and testing code out in industry, and for school projects, because errors could potentially lead to safety hazards and loss of revenue. To avoid technical debt as a practitioner I will develop applications that require little to no maintenance, and also that are as functioning a possible on the first revision. Examples of said claims can be the projects I submitted for this class, as well as the projects I have submitted for other classes illustrating programs which limit technical debt.

**References**

Aebersold, K. (2021). Functional vs. Non-functional Testing. SmartBear. https://smartbear.com/learn/automated-testing/software-testing-methodologies/

Arnold, D. (2000, August 23). The Explosion of the Ariane 5. UMN. https://www- users.cse.umn.edu/~arnold/disasters/ariane.html

Hamilton, T. (2021, October 8). Unit Testing Tutorial: What is, Types, tools & Test EXAMPLE. Guru99. https://www.guru99.com/unit-testing-guide.html