Jonathan Campos

CS-320-T3331 Software Test Automations and QA 23EW3

Professor Kraya

Summary

Before starting each assignment, I ensured I had a good understanding of the software requirements that needed to be fulfilled. Reviewing Milestone three one of the requirements of the class was the following:" The contact object shall have a required unique contact ID string that cannot be longer than 10 characters. The contact ID shall not be null and shall not be updatable." In the test class I wrote the following line which asserts the ID requirement. *Contact contact = new Contact("1234567890", "Jonathan", "Doe", "1234567890", "12345 November Drive"): assertEquals(10, contact.getContactId().length());*It defines a public void method called testContactIdLength for the Junit test class. The code calls the assertEquals() method to check that the length of the contact ID is less than or equal to 10. If the ID was over 10, the test would fail. The contact test coverage was at 100 percent, which means every line of code was at least executed once. I understand that 100 percent code coverage doesn’t mean my tests were comprehensive; however, as I moved to the next milestone, I was able to learn and think of tests that were more comprehensive. A requirement of the Contact Service class was the following: The contact object shall have a required firstName string field that cannot be longer than 10 characters. The firstName field must not be empty.I coded the following snippet to ensure it met the requirement. If (firstName == null || first.name.Length() > 10) a new IllegalArgumentException ("Invalid first name") is thrown.In the Junit, I applied the assertNotNull method to test if the string is valid. I did, however, test the firstName length in the contact class rather than the Junit code, which I now realize was a mistake.In the taskTest, I created a method called testTaskContructor(), which created a "Task" with the following arguments: "1" for the ID field, "Task" for the name field, and "Run" for the description field. I used three assertEquals statements to check if the Task instance has the expected values, and if it did not, the test would fail. Reviewing the code now, I realized I did not test if the character restrictions on the arguments applied, which is something I would now be able to correct.

Throughout this project, I learned a few software testing techniques. In order to meet the business criteria of the project, I first used function testing to validate that the methods behaved as expected. An example would be to test if the task id was no longer than 10 characters. I then applied unit testing using Junit to ensure the test failed if the ID was longer than 10 characters. As the project's sole developer, I was also able to use white box testing. White box testing investigates the internal data structures, internal design, code structure, and functioning of the software, as opposed to black box testing, which only concentrates on functionality. I wrote the following code in the appointmentTest: @Test(expected = IllegalArguementException.class) public void testInvalidAppointmentID() new Appointment(null, new Date(), "Dentist appointment"). This test checks whether the constructor of the Appointment Class throws an IllegalArgumentException when an ID is null. There are a few other testing techniques I was not able to perform. Black box testing is a "method of software testing that examines the functionality of an application without peering into its internal structures or workings." This method of testing can be applied to virtually every level of software testing: unit, integration, system, and acceptance. Due to not having to create a user interface, I was not able to verify the function of entering values for the class parameters. Concerning applications and implications, black box testing is very useful in ensuring that every function exhibits the desired behavior and meets stakeholder expectations. It is frequently a prerequisite of a project. Testing mobile and online applications would be necessary to find bugs and enhance the user experience. White-box testing makes sure the developer can cut down on errors and write code that complies with stakeholders' demands. Also, it might make the document easier to understand and maintain for other members of the organization. This was the first time I ever applied any type of testing to my code. I always assumed that a static test would suffice and that I would be able to create functional applications by using numerous if-else statements. With every new test case, I began to understand the need to exhaust every possible scenario, especially when working for a client. I can understand how easy it can be to simply not write test cases and assume the application will not break with any input you enter. I understand that it is also important to appreciate the complexity and interrelationships of the code being tested, as they can impact the effectiveness of the testing process in a software development lifecycle. If one is not considerate of the testing process, the design of the test cases could be overlooked and contain defects and bugs that will cost the company money to debug and resolve. I believe a developer should be able to apply functional testing to their own code. I can understand how one can be biased when writing their test cases to ensure 100 percent coverage. I attempted to combat this by defining clear objectives for each test. This allowed me to always think about the software requirements as I created each test case. In the earlier milestones, I created a junit test with very few test cases because I assumed that as long as all the test cases passed, my code was thorough. I can understand not being disciplined on personal, fun projects you do on the weekends, but when it comes to creating software that will be utilized by many users, you cannot cut any corners. Very few companies with complex processes are able to catch and fix issues in the code you present. When working with a startup, they may not have the luxury of having a QA team. They rely on your software skills and thoroughness to ensure you meet their business requirements, and this is a responsibility that should not be taken lightly. I plan on conducting as many testing techniques as possible when I finally become a software developer. Techniques such as static reviews and peer-to-peer reviewing can greatly impact the quality of your code.

References

* *Editor, C. S. R. C. C. (n.d.). Black box testing - glossary: CSRC. CSRC Content Editor. Retrieved February 23, 2023, from https://csrc.nist.gov/glossary/term/black\_box\_testing*
* *GeeksforGeeks. (2022, November 29). Software engineering: White box testing. GeeksforGeeks. Retrieved February 23, 2023, from https://www.geeksforgeeks.org/software-engineering-white-box-testing/*