Janice Ainembabazi

Southern New Hampshire University

CS-320 Software Test, Automation QA

Robert Tuft

October 20th 2024

**Summary and Reflections Report**

**1. Summary**

**a. Unit Testing Approach for Each Feature**

* **Contact Feature**:
  + I tested the Contact feature by checking that all the information entered, like names and phone numbers, was valid. For example, I wrote a test called testFirstNameCannotBeNull to make sure the first name is not left blank. This matched the requirement that the first name cannot be empty and must be 10 characters or less.
  + E.g.: In my test file, I used Assertions.assertThrows(ValidationException.class, () -> new Contact(null, validFirstName, validLastName, validPhone, validAddress)); to make sure a validation error happens when the first name is empty.
* **Task Feature**:
  + For the Task feature, I wrote tests to make sure the id, name, and description fields followed the rules. For instance, in the test testIDLongerThan10CharactersFails, I checked that if an ID is longer than 10 characters, the code throws an error. This test matched the requirement that IDs must be short.
  + E.g**.**: I wrote Assertions.assertThrows(ValidationException.class, () -> new Task("too-long-id", name, description)); to verify that long IDs are not allowed.
* **Appointment Feature**:
  + When testing the Appointment feature, I focused on checking that the appointment date was in the future and that the description wasn’t too long. My test testAppointmentDateCannotBeInThePast made sure that if someone tries to set an appointment in the past, an error occurs.
  + E.g.: Assertions.assertThrows(ValidationException.class, () -> new Appointment(validId, pastDate, validDescription)); made sure the code stopped past dates, which was the requirement.

**b. Quality of JUnit Tests**

* I know my JUnit tests are good because they covered a lot of different situations. I tested both what happens when the information is right and what happens when it’s wrong. This helped make sure that all parts of the code were checked.
* **Example**: In ContactServiceTest, I tested methods like addContact, deleteContact, and updateContact to make sure they all worked as expected. By testing every field (like firstName), I could be sure everything worked correctly.

**c. Writing JUnit Tests**

* I made sure my tests were technically sound by checking each rule one by one. For instance, I used Assertions.assertNotNull(task) in TaskTest to confirm that the task object is created properly and isn’t empty.
* I also worked to make my code efficient. I used setup methods like @BeforeEach in ContactTest to set up the test data only once and reuse it, saving time and reducing repeated code.

**2. Reflection**

**a. Testing Techniques**

* **Unit Testing**:
  + I used unit testing to check small parts of the code, like individual functions. This testing is fast, and it helped me find issues early because I tested each part on its own.
* **Static Testing**:
  + I also checked the code without running it, using tools in the code editor to spot mistakes, like typos or missing variables.

**b. Other Testing Techniques**

* **Integration Testing**:
  + This kind of testing checks if different parts of the program work well together. It’s used when you want to make sure that components, like databases and APIs, are connecting properly.
* **System Testing**:
  + System testing looks at the entire program to see if everything works together, from start to finish. It’s useful when you’re ready to check if the whole system meets the user’s needs.
* **Regression Testing**:
  + This type of testing repeats older tests to make sure that when new features are added, they don’t cause new problems. It’s great when making updates to a system so that older features still work.

**c. Practical Uses of Testing**

* **Unit Testing**:
  + Unit testing is useful when you are building small parts of a program and want to make sure each part works before combining them into a bigger system.
* **Integration Testing**:
  + This is important when building a system that connects with other programs, like a bank app that connects to a server.
* **System Testing**:
  + System testing checks everything from beginning to end, so it’s good for testing the whole system before it’s given to the user.
* **Regression Testing**:
  + This testing helps make sure that new features don’t cause bugs. It’s useful when you keep adding new parts to a program.

**d. Mindset as a Tester**

**i. Being Careful**

* I was very careful to read each requirement and make sure my tests matched what was needed. For example, when testing the Appointment feature, I paid close attention to how the dates worked and tested different scenarios, like past and future dates.
* Understanding how different parts connected helped me see how important it was to make sure all parts fit together properly.

**ii. Limiting Bias**

* I tried to think about the tests as if I were another developer or a user, not just myself. For example, I wrote tests to check what would happen if an ID was left blank, even though I usually expect IDs to be filled.
* E.g.: In ContactServiceTest, I included a test called testDeleteNonExistentContactDoesNothing to make sure that trying to delete something that doesn’t exist wouldn’t cause an error.

**iii. Commitment to Quality**

* It was important to test all possible scenarios, even if they seem unlikely. For instance, testing every field separately in ContactTest helped me make sure everything worked as expected.
* **Avoiding Technical Debt**:
  + To avoid problems in the future, I plan to use tools like continuous integration (CI), which can automatically check my code for errors every time I make a change. This helps catch bugs early.
  + E.g**.**: In future projects, I will set up automated tests that run every time I make an update, so I know right away if something breaks.

**REFERENCE**

Kent, B. (2003). *Test driven development*. Google Books. <https://www.google.com/books/edition/_/zNnPEAAAQBAJ?hl=en&gbpv=1&pg=PR7&dq=Beck%2C%2BK.%2B%282003%29.%2BTest-Driven%2BDevelopment%3A%2BBy%2BExample.%2BAddison-Wesley>.

Morgan, P., Samaroo, A., Thompson, G., & Williams, P. (2019). *Software testing : An istqb-bcs certified tester foundation guide - 4th edition*. BCS Learning & Development Limited.

Myers, G. J., Sandler, C., Badgett, T. (2011). The Art of Software Testing. Germany: Wiley.