Tonessa V. Chatten

CS 320 Software Testing:

Summary and Reflections Report

Summary

Unit testing is essential for developing high-quality software. It helps to reduce the risk of bugs and improve the quality of the code by quickly identifying and fixing defects. I applied a similar unit testing approach to the three features, ensuring the tests aligned with the software requirements. For example, the Contact service had some specific requirements:

- The Contact object must have a unique contact ID string that is no longer than 10 characters and cannot be null or updated.
- The Contact object must have a firstName and lastName field that are no longer than 30 characters and cannot be null.
- The Contact object must have a phone field, a string that is exactly 10 digits and cannot be null.
- The Contact object must have an address field, a string that is no longer than 30 characters and cannot be null.

I wrote unit tests to verify that the Contact object met these requirements. For example, I wrote a test to ensure that the Contact object could not be created without a contact ID string. I also wrote a test to ensure that the contact ID string could not be updated.

Unit testing is a valuable tool for ensuring the quality and reliability of software. By following industry standard best practices for unit testing, I was able to identify and fix defects in

the Contact service early, before they could cause problems for users. The Contact service also includes crud operations for Contact objects. These operations were specific requirements for one of the three features, but all had similar requirements. Therefore, the software must be implemented with those requirements to pass the unit tests. I tested each field and operation to ensure the validity of the test cases. Overall, the quality of my unit tests was good.

Overall, I feel pretty good when it comes to JUnit testing. I do feel like I need to learn more on it, but overall, have it down pact.

```
import static org.junit.jupiter.api.Assertions.*;

class ContactTest {
//test ContactID
   @Test
   void testContactIDTooLong() {
      Assertions.assertThrows(IllegalArgumentException.class,() ->{
         new Contact("1234567890000","Janet","Harris","7045555555","678 Fake St Charlotte");
      });
}
   @Test
   void testContactIDNull() {
      Assertions.assertThrows(IllegalArgumentException.class,() ->{
         new Contact(null,"Janet","Harris","7045555555","678 Fake St Charlotte");
      });
}
   //test firstName
   @Test
   void testFirstNameTooLong() {
      Assertions.assertThrows(IllegalArgumentException.class,() ->{
         new Contact("1234567890","JanetRogerTom","Harris","7045555555","678 Fake St Charlotte");
      });
}
   @Test
   void testFirstNameNull() {
      Assertions.assertThrows(IllegalArgumentException.class,() ->{
         new Contact("1234567890",null,"Harris", "7045555555","678 Fake St Charlotte");
      });
}
   //test lastName
   @Test
   void testLastNameTooLong() {
      Assertions.assertThrows(IllegalArgumentException.class,() ->{
         new Contact("1234567890","Janet","HarrisSmithRogers", "7045555555","678 Fake St Charlotte");
      });
}
   @Test
   void testLastNameNull() {
      Assertions.assertThrows(IllegalArgumentException.class,() ->{
         new Contact("1234567890","Janet",null, "7045555555","678 Fake St Charlotte");
      });
}
   //test phoneNumber
   @Test
   void testPhoneNumberTooLong() {
      Assertions.assertThrows(IllegalArgumentException.class,() ->{
         new Contact("1234567890","Janet","Harris","70455555556","678 Fake St Charlotte");
```

With the lines of code above, you can see that I was checking for different scenarios. I had created tests on whether the phone number was too long and if the last name was null.

Reflection

I used dynamic testing, a software technique that involves testing the dynamic behavior of software code. This technique allowed me to check the behavior of dynamic variables, which are not constant, and find weak areas during software runtime.

I did not use static testing, another software technique that involves testing code without executing it. Static testing can identify defects that dynamic testing cannot easily find, such as development standard breaches and dependencies and inconsistencies in software models.

Regarding mindset, I adopted a cautious approach to testing the various cases, ensuring that I captured all the test cases. It was important to appreciate the complexity and interrelationships of the code, as each of the class objects had specific requirements that must be fulfilled to certify the code as working correctly. This required a deep understanding of what each part of the code was doing.

Bias can be a major factor in code review, as people may be confident in their code and, therefore, less likely to test it thoroughly. For example, when creating variables in a class, you may be confident that the code is effective because you created it yourself. However, we should all try to limit bias regarding software testing.

Discipline is essential for software engineers who want to ensure the quality of their code. It helps prevent bugs and errors, saving millions of dollars and even lives. It is important to

avoid cutting corners when writing or testing code. For example, testing the Contact service to ensure that each contact has a unique ID that cannot be updated helps to prevent errors and ensure that each contact is unique and cannot be interchanged with another contact.