Charlie Vinson

CS-320

10/15/2023

Summary

1. **Describe your testing approach for each of the three features.**

My testing approach for these three features of the application included, first, identifying the requirements outlined by the client. Once these requirements are identified, and the data that should not be allowed by the system is defined, I created test cases that include this type of data. Successful completion of these tests requires that all of this bad data is caught by the system and that the user is not allowed to proceed until good data is given to the system. This process includes checking each parameter of the classes being created to ensure that they are appropriate, as well as checking this data each time that objects are updated by the system. If a parameter is found to be of an incorrect format, an error should be thrown by the system. An example of this from the Contact feature of my application can be found in the ContactTest.java file. When objects of the class Contact are created, the ContactTest class checks if an error is thrown by the system, which should occur when the requirements are not met. Then, when objects of the class Contact are updated, they are checked again to ensure that the updated data also meets the requirements. Because these tests are designed specifically to check the relatively simple requirements outlined by the client, I believe that my approach was completely aligned to the software requirements and that it successfully verified that the software meets these requirements.

1. **Describe your experience writing the JUnit tests.**

My experience writing the JUnit tests included the previously described process for developing the specific tests, then ensuring that the coverage of these tests met the goals of the client (at least 80%). While most aspects of my application tests were technically sound and covered the system almost entirely, the Contact and ContactService classes needed improvement to reach this goal. This improvement included further verification that updates to the system resulted in additional checks of the data. For example, under the test labeled testContactFirstNameNull, which tests objects of the Contact class to verify that a null entry for the firstName parameter results in an error being thrown, an additional check was included. This involved creating an object with good data for the firstName parameter, then attempting to update this same object with a null firstName. The test then verifies that an error is thrown and that the object’s firstName has not been updated by checking the firstName parameter of the object. As a result, I believe that my code is both technically sound and efficient.

Reflection

1. **Testing Techniques**

The Contact, Task, and Appointment classes of the application utilize unit testing to ensure that their functions, and the functions associated with their relative service classes, obey the rules defined by the software requirements. To ensure that these unit tests offered the greatest coverage, the software requirements were examined to determine ways which data can be considered to be outside of the requirements. Because the requirements of this application were relatively simple, this type of data can be easily defined. For example, the requirements specify that objects of the Contact class cannot have an ID, firstName, or lastName attribute that is either null or longer than 10 characters. Similarly, an object’s phone attribute must be exactly 10 characters and an object’s address field cannot be null or longer than 30 characters. To develop unit tests for these requirements, I attempted to create objects with null attributes or String type attributes of 50 characters to verify that these objects would not be created by the application. One aspect of this that I failed to consider is the data type of the attribute. My unit tests only attempt to create objects using either null or String parameters, and they do not attempt to create objects using integers or Boolean values. Improving the tests in this way is important because this type of logic can be applied to a wide array of other types of applications, which may inadvertently receive data of these incorrect types. In these situations, it is important to verify that the system correctly rejects this data, as well.

1. **Mindset**

As a software tester, it is important to employ caution when developing tests to verify software requirements. While some methodologies promote popular best practices, such as limiting the use of repeating code, software testers often have conflicting guidelines for writing good code that are generally more important to follow throughout the testing process. Similarly, it is important to avoid including bias when reviewing code. While code may seem to be secure given the expected or what may be considered predictably unexpected input, it is necessary to consider as many different types of input as possible given the circumstances or software requirements. Both of these qualities of a software tester require being disciplined and maintaining good habits as a software professional, overall.