CS320 Project Two - Summary and Reflections Report

Kari L. Cheslock - 06/11/21

**Summary**

My testing approach for the three code modules that I created for this project was to essentially go through the code line by line and test each function or method individually and with multiple values, including edge cases. I used the software requirements as a guide to design the test cases I would need. For example, when the requirements said that the contact’s phone number should be exactly 10 digits and not null, I checked each of these three aspects. Specifically I checked that the length of the phone number field was exactly length ten, not null, and contained only digits. As an example, one of the JUnit tests for the phone number field is shown below.

void testPhoneWrongLength() {

Assertions.assertThrows(IllegalArgumentException.class,() -> {

new Contact("d84f22182b", "Jane", "Doe", "1234567890123", "123 Main Street");

});

}

After setting up and running all of my JUnit tests, I determined that they were effective by first checking that all of them passed without error and then by checking the coverage percentage for each of the test classes. For each of the test classes I created, the coverage was over 80% which gave me confidence to say that my tests that I designed showed that my code was working as intended.

I ensured my code was technically sound by running many different values through the JUnit tests, although only one value was illustrated in the final submission. I also checked my code and my JUnit tests against the software requirements ensuring that all expected fields were there and that the methods I created worked as expected. For example, it was important for each of these code modules to have a unique identifier for the object being created. For each method that added a new object, I inserted code that checked if the unique identifier already existed which would then throw an exception if it returned true. Otherwise, it would create a new object. The code for creating a new task service is illustrated below, which includes the code to check for a unique identifier.

public void newTask(String taskId, String name, String description) {

if (searchId(taskId) != null) {

throw new IllegalArgumentException("Task Id already exists");

} else {

Task task = new Task(taskId, name, description);

taskList.add(task);

}

}

I ensured my code was efficient by limiting repetitive code in the various modules. As an example, the code that was needed to update various fields started by searching the list of objects for the given unique identifier. Rather than having code in each individual update method to search the array of objects by unique identifier, I wrote one method to search the array and called this method in each of the update methods. The search method for the contact service class is illustrated below.

public Contact searchId(String contactId) {

Contact contact = null;

for (int i = 0; i < contactList.size(); ++i) {

if (contactList.get(i).getContactId() == contactId) {

return contactList.get(i);

}

}

return contact;

}

**Reflection**

The software testing technique that I used for each of these code modules was unit testing using JUnit tests. Unit testing is done on each individual unit of code and is used to demonstrate that these individual pieces work as intended. For example, with this project, I tested methods to create objects, delete objects, and update fields within each object. I also tested unique identifiers to ensure they were unique and that incorrect inputs would throw exceptions as expected.

The software testing techniques that I did not use were integration testing, system testing, and acceptance testing. Integration testing involves testing the software after integrating the individual pieces of code to ensure that the code still functions as expected after being combined. System testing involves testing the entire software package end to end to ensure overall proper functionality of the software in its entirety. Acceptance testing involves testing the software against the customer’s quality assurance requirements to show that the software meets specifications. All of these types of testing work together to ensure that the finished product is of the highest possible quality for the customer.

All of these testing techniques have practical applications in the software development life cycle since they represent a natural flow of testing software from the beginning of the development life cycle all the way up to the final testing prior to delivering the final product to the customer. Testing starts with unit testing which is typically done directly by the developer as the code is being written and verifies that the code works as expected and throws exceptions when unexpected values occur. Next is integration testing which shows that individually pieces of code can be integrated seamlessly and still work without error. System testing is the next logical step and can typically be done by a team of testers, particularly at a larger company. This stage shows that the software as a package works from beginning to end. Finally, acceptance testing is the final quality check on the finished product and can be used to demonstrate to the customer that their business requirements have been met.

Looking back at the mindset I adopted throughout this project, I employed caution by making sure that I strictly followed the requirements as they were written and did not attempt to read between the lines to write any extra code that was not requested. Knowing that the code was intended to be part of a larger project, I felt it was important to stick to the described scenario without going any further. If I had added any extra code or additional methods, they may not have been compatible with other units that were still to be developed or they may have gone against the customer’s requirements.. As an example, I could have been proactive with the contact class and added additional fields that may be needed, such as multiple phone numbers or an email address, but that may have caused issues with the final product.

I tried to limit bias in my code by being as consistent as possible from module to module. I also tried to be as consistent as possible in the testing of my code. I tested every possible aspect of the code that I could think of, keeping the listed requirements in mind the entire time. Bias could be a concern for a developer testing their own code since it is difficult to see errors or other issues in code that they wrote themselves. If possible, code should always be tested by someone other than the developer to ensure nothing was missed and all requirements are met.

Lastly, being disciplined when developing code is of the utmost importance in terms of developing quality code. In order to produce code that is of the highest quality, code must be written fully and completely without cutting any corners. Specifically, each requirement must be checked to ensure that it was met and each aspect of the code should be tested to ensure that it behaves as expected. Being disciplined and taking a structured approach to software development is the best way to ensure that this happens. Developing personal standards for code quality is perhaps one of the best ways to remain disciplined throughout any project that a developer works on. If a developer is disciplined and sticks to the strictest code quality requirements, the finished product delivered to the customer has greater assurance to be a high quality product.

Reference

Hambling, B., et al. (2015). *Software Testing - An ISTQB-BCS Certified Tester Foundation Guide (3rd Edition).* BCS The Chartered Institute for IT. Retrieved from

https://app.knovel.com/hotlink/pdf/id:kt00UC2FT1/software-testing-an-istqb/front-matter