Jordan Matsumoto

CS 320

February 20, 2021

Project Two

While coding the contact and task service projects, my testing approach was aligned to the software requirements by ensuring that both the class and service requirements were address in each project. For example, one of the requirements for the contact class is that an object cannot have contact ID string that is greater than 10 characters. This is represented in the code from valid and invalid ID’s in the lines “protected final String validID = "123456789";” and “protected final String badID = "1234567890a";.” Note the second string is invalid because it contains 11 characters and does not meet the requirement. Through ensuring that the requirements are correctly tested for with JUnit tests, I can defend the quality of the code based on the coverage percentage. For example, the coverage percentage of the JUnit testing performed in the task service project shows us that all tests successfully passed and therefore the requirements are met. On the other hand, a low coverage percentage would signify that not all of the requirements were coded properly and therefore additional revisions would be required to ensure the project was coded properly.

I ensured that the code was technically sound by using comments to highlight specific elements of testing such as checking character length and the ability to add, delete, or update contacts. The following lines of comments and code represent the stability of the code by further ensuring the requirements are met:

// Verify that the contact object we created has the specified parameters

@Test

public void verifyContact() {

// Just assert that our goodContact meets all of our fields

System.out.println("Checking Contact constructor");

assertEquals(goodContact.getId(), validID);

assertEquals(goodContact.getFirstName(), validFirstName);

assertEquals(goodContact.getLastName(), validLastName);

assertEquals(goodContact.getPhoneNumber(), validPhoneNumber);

assertEquals(goodContact.getAddress(), validAddress);

Lastly, I ensured the code was also efficient by ensuring that testing verified an invalid ID as an error. This can be seen through the following lines of comment and code:

// Verify that setting the ID to an invalid one raises an AssertionError

@Test

public void verifyBadId() {

System.out.println("verifying id...");

rule.expect(AssertionError.class);

new Contact(badID, validFirstName, validLastName, validPhoneNumber, validAddress);

While coding the service milestones for Modules Three, Four, and Five, the software testing technique that I employed were JUnits that ensured requirements were met. Each milestone contained software tests to check that ID strings were not longer than a specific amount of characters and that input sections were not null. Additionally, the tests confirmed that tasks, contacts, and appointments could be added and deleted based on their specific, unique ID. The majority of these JUnit tests contained functional testing elements because these tested the software against the requirements of the milestones. Similarly, adding and deleting information relied on integration testing to ensure that specific functions of the project were properly coded to output the correct results.

On the other hand, system testing has not yet been utilized as it requires the three milestones and completed, integrated system to ensure that all requirements are met for the finished project. Additionally, acceptance testing is another phase of functional testing that will be used to ensure the final project is in compliance with requirements and the end user’s experience will be as intended. For this type of software testing, external testers will be required to perform beta testing and provide feedback from their experience. Performance testing will also be required for the majority of large projects as it ensures that the demand of the system will not outweigh the capabilities of the user’s computer. Security testing also plays a crucial role as a non-functional testing technique to find possible loopholes that attackers could use to retrieve personal information. These additional testing techniques all ensure that final project is capable of providing the end user with the intended experience without putting too much stress on their computer and ensuring both the company and individual’s information is secure from hackers. While smaller projects may not require all of the testing techniques mentioned above, it is important as a developer to identify what testing should be implemented and when it should occur. For example, testing the security of the system would not be beneficial if it was implemented after only one unit of project was completed.

While working on this project, the mindset that I adopted was to carefully read each line of code after writing it and double check references to ensure that the correct Java triggers were implemented and the output was correct. I further employed this element of caution by ensuring that the JUnit tests were coded correctly and each requirement was not only coded correctly but also tested for. In order to ensure that these requirements were not only coded correctly but also tested for correctly, I had to verify that the possible inputs, outputs, and tests were formulated in a way that represented their interrelationship. For example, the following lines of code verify that the appointment object contains the correct fields:

@Test

public void verifyAppointment() {

System.out.println("Checking Appointment constructor");

System.out.printf("\tChecking that the id is %s\n", validID);

assertEquals(goodAppointment.getId(), validID);

System.out.println("\tChecking that the date has the correct fields");

Calendar calendar = Calendar.getInstance();

calendar.set(validYear, validMonth, validDay);

Date date = calendar.getTime();

Similarly, bias was limited in review of the code by creating tests that checked for a wide array of possible inputs to ensure if they would uphold requirements by remaining within a certain character length, not contain null inputs, and have the ability to be added, deleted, or edited. Bias would have been integrated into the testing if they JUnit tests were coded in a way that specifically aligned with how the contact, task, and appointment objects were coded. Therefore, not only is it important to eliminate bias from tests when the developer is also the tester but to also remain disciplined when writing objects or tests. Writing code that correctly analyzes all requirements for a project is vital for ensuring that bugs, hacks, and function are not compromised once the final system is released. Technical debt can be avoided by providing clear comments on the specific functioning and testing that each line or segment of code offers. This not only provides a strong reference for ensuring integration between the coding and testing of requirements but also allows other team members to identify what should be tested if someone else reviews or finishes the project.

References

Hambling, Brian Morgan, Peter Samaroo, Angelina Thompson, Geoff Williams, Peter.

(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/toc/id:kpSTAIST01/software-testing-an-istqb/software-testing-an-istqb