Project 2

[CS-320-T5615 Software Test Automation& QA 22EW5](https://learn.snhu.edu/d2l/home/1069796)

Valerie J. Smith

June 17, 2022

Project 2 Summary

Describe your unit testing approach for each of the three features.

The unit testing approach for each of the features, Contact Service, Appointment Service, and Task Service, centered around creating JUNIT tests that primarily focused on assertions around the main project requirements.

Each of the three features contained an object class that held the object and validations, as well as a service class that imported the object class in order to use the class in various methods, such as update, remove, and to add the objects to an Array List. Each class had a corresponding JUNIT test class that contains tests for each specific class.

The JUNIT plugin for maven was used to import TestMethodOrder, DisplayName, and Order annotations, which helped to run the unit tests in succession. The test cases for each class ensured that the functionality that was expected in the requirements was tested to ensure that the code would function as expected.

To what extent was your approach aligned to the software requirements? Support your claims with specific evidence.

To ensure that my testing approach aligned to the software requirements, I chose a unit test strategy that would test for the specific requirements that were listed, such as the variable length and whether or not a variable was to be null.

For example, the contact class requires that a first name can be no longer than 10 characters. In a suggested code sample, the instructor provided the sample that if the first name was more than 10 characters to just get the substring that was 10 characters. I felt this approach was intuitive and prevent an exception from being thrown, so I decided to use it. This code also sets the first name to the string NULL if the user enters null.

To test that the first 10 characters would only be accepted by the program, I created a unit test for character validations. The test creates a new contact with the same parameters as the Contact object. The first name parameter in this case is too long. I then use the junit assertEquals to test that the program will only accept the first 10 characters by asserting this to be true.

void testCharacterValidations() {

Contact contact3 = new Contact("ValerieJeanSmith", "SmithMillerValo", "11234567899999", "myextremely long address greaterthan 30 characters, usa");

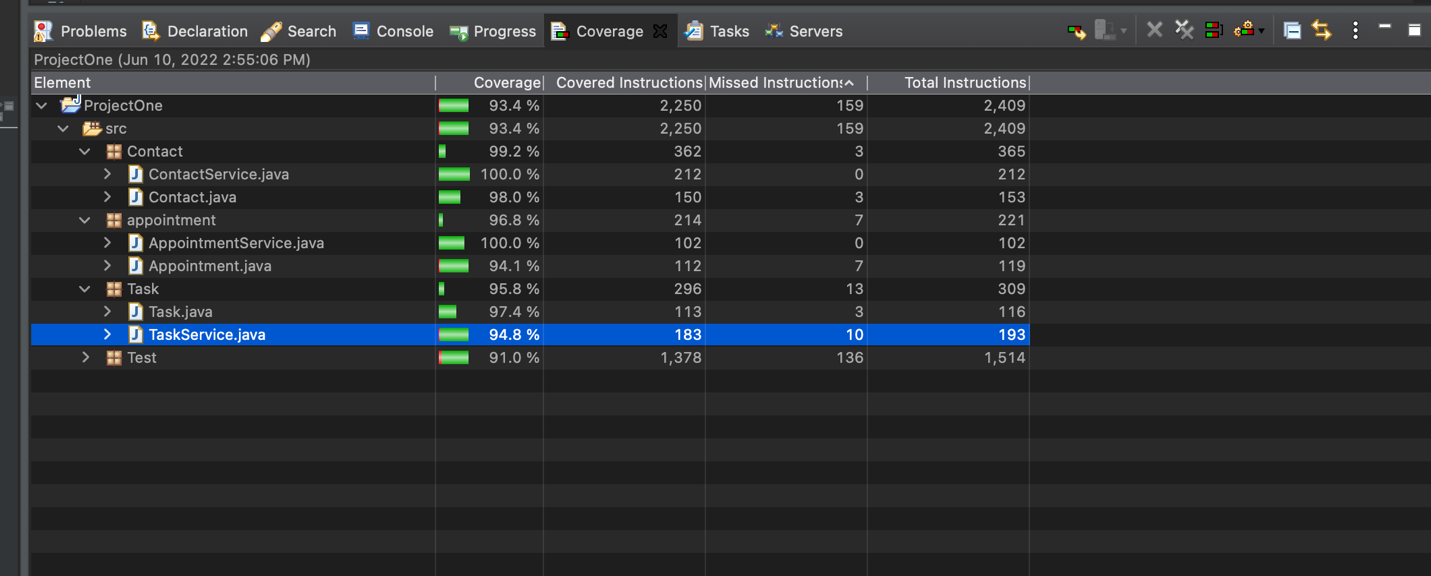
assertEquals(true,contact3.getFirstName().equals("ValerieJea"));

}

Defend the overall quality of your JUnit tests. In other words, how do you know your JUnit tests were effective based on the coverage percentage?

The overall code coverage was found by running the program as coverage-> junit from eclipse. In order to see the overall coverage with both all test files, I used the @Order annotation from the OrderNotation class. For the contact, the task, and the appointment assignment, the code coverage is over 90%. Once the coverage report is run, the individual files show where there might be something that isn’t covered. The validations that are required by the assignment are all covered in the unit tests, which assures me that the programs will work as expected. The code coverage report generated shows me that over 90% of the code that was written has been successfully tested.

This is a screen shot of the overall code coverage:



Describe your experience writing the JUnit tests.

How did you ensure that your code was technically sound? Cite specific lines of code from your tests to illustrate.

To ensure that my code was technically sound, the classes for Contact.java , Task.java , and Appointment.java contain all of the variables that are required by the assignment as well as their getters and setters. To ensure consistency when creating a new instance of the any of the objects, there is a constructor that has been created for each class. This will ensure that each object instance is created only as required by the assignment. The test will also ensure that the code is technically sound by meeting the requirement that each contact gets a unique user id. The following example is from the Contact class:

void testContactClass() {

Contact contact = new Contact("Valerie", "Smith", "1123456789", "my address, usa");

assertEquals(true,contact.getFirstName().equals("Valerie"), "Contact first name should be Valerie");

assertEquals(true,contact.getLastName().equals("Smith"), "Contact last name should be Smith");

assertEquals(true,contact.getNumber().equals("1123456789"), "Contact phone number should be 1123456789");

assertEquals(true, contact.getAddress().equals("my address, usa"), "Contact address should be my address, usa");

// test toString method

String myContact = contact.toString();

assertEquals(true, contact.toString().equals(myContact));

contact.setFirstName("Ben");

assertEquals(true,contact.getFirstName().equals("Ben"), "Contact name should be now set to Ben");

contact.setLastName("Miller");

assertTrue(contact.getLastName().equals("Miller"), "Contact name should now be set to Miller");

contact.setNumber("2222222222");

assertTrue(contact.getNumber().equals("2222222222"), "Contact phone number should now be set to 2222222222");

contact.setAddress("Bens address");

assertTrue(contact.getAddress().equals("Bens address"), "Contact address should now be set to Bens address");

contact.getContactID();

assertEquals(true, contact.getContactID().contentEquals("0"), "The contact ID should still be 0");

assertEquals(false, contact.getContactID().contentEquals("1"), "The contact ID should not be 1");

}

How did you ensure that your code was efficient? Cite specific lines of code from your tests to illustrate.

In order to ensure that the code is efficient, the code is abstracted into two different files. For all three assignments, there are separate files for the object class as well as a file that has a service class. The service class contains helpful methods in order to perform functions on the instantiation of the object class members.

From the contact assignment, the contact service is tested to ensure that it can create an array list of one or more contacts. The contact information needs to pass validators or it will not be added to the list. This is an example of efficiency as the code only allows the correct information to be added to the list. This is an example from the Contact Service:

void testContactServiceClass() {

Contact contact = new Contact("Valerie", "TestName", "8145455545", "123 Test Lane");

Contact contact2 = new Contact("GamoraStarlord", "TestName", "8145455545", "123 Test Lane");

ContactService service = new ContactService();

service.addContact(contact);

service.addContact(contact2);

assertEquals(true,service.contactList.get(0).getFirstName() == "Valerie", "Contact First Name should be Valerie");

assertEquals(true,service.contactList.get(0).getLastName() == "TestName", "Contact Last Name should be TestName");

assertEquals(true,service.contactList.get(0).getNumber() == "8145455545", "Contact phone number should be 8145455545");

assertEquals(true,service.contactList.get(0).getAddress() == "123 Test Lane", "Contact address should be 123 Test Lane");

assertEquals(true, service.contactList.size() == 2);

}

From the Task Service assignment, there is an update method that will update the task name by using the task ID. If the taskID is found in the array list, then the method will call the object’s setFirstName method to update the new first name. This function is an example of efficiency, as it is searching the array list for the taskID as well as updating the new task name that is also passed in. The Task class contains the validation and will validate the new task name that is passed in. The following example is from the Task Service, and similar implementations are found in the Contact as well as the Appointment Service:

TaskService service2 = new TaskService();

service2.addTask("my third task", "these are the items for the third task");

service2.addTask("my fourth task", "these are the items for the fourth task");

assertEquals(true, service2.taskList.size() == 2,

"Task list should contain 2 elements");

service2.displayTaskList();

service2.updateTaskName("Online Shopping", "3");

service2.displayTaskList();

assertEquals(true, service2.taskList.get(1).getName() == "Online Shopping",

"Task Name should be Online Shopping");

}

Project 2 Reflection

Testing Techniques

What were the software testing techniques that you employed in this project? Describe their characteristics using specific details.

The software testing techniques that were used for each of the milestones were utilized by creating JUNIT tests. JUNIT tests are unit tests specific to the Java programming language. Unit testing itself is a “form of white box texting, in which test cases are based on an internal structure (Junit tutorial with examples: Setting up, writing, and running Java Unit tests 2022)”. JUNIT testing is also considered a form of regression testing (Junit - Test Framework).

With unit testing, the tester chooses what to test based upon the functionality of the code. It is a goal of unit testing to test the least amount of code as possible. (Junit tutorial with examples: Setting up, writing, and running Java Unit tests 2022)

With all three milestones, it was possible to test small parts of code by abstracting the code into two classes, the object class and the object service class. The contact milestone contained the contact and contact service class; the task milestone contained the task and task service class; and the appointment milestone contained the appointment and the appointment service class. Each class contained a test file that was used to test the functionality contained in each class. For object classes, the getters and setters are tested. For object service classes, the functions that were specified as necessary to fulfill the requirements were tested, such as adding or deleting an item from a list.

Software testing techniques used to set up the unit tests included creating the application as a maven application which imported the required JUNIT 5 dependencies. By importing the JUNIT plugin with Maven, I had access to the TestMethodOrder, DisplayName, and Order annotations. Using these features for unit testing was a way to keep the tests more readable and enabled me to run all of the tests in order.

@TestMethodOrder(OrderAnnotation.class)

public class AppointmentTest {

@Test

@DisplayName("Test to Get and Set Appointment Information.")

@Order(1)

void testAppointmentClass() {}

Another software testing technique was to name the test method as intuitively as possible, such as testAppointmentClass(), where I was able to test the getter and setter functionality of the object classes, and the methods of the service object, such as add, delete, or update.

Appointment appointment = new Appointment(date4, "my first test appointment");

assertEquals(true, appointment.getDate().equals(date4), "The date should be the current date");

assertEquals(true, appointment.getDescription().equals("my first test appointment"), "Description should be my first appointment");

The software testing technique to create unit tests focused on fulfilling the requirements that were gathered from the assignment, such as a contact name that could only contain a maximum of 10 characters. It was also possible with the unit testing to test the constructor for each object class. The unit tests that were completed on each milestone will assure that the original requirements remain intact in case of future code updates, as well as provide documentation to another developer about the formation of the code.

What are the other software testing techniques that you did not use for this project? Describe their characteristics using specific details.

A software testing technique that I did not use for the milestones was mocking, which “helps to isolate units of code to focus on validation (Junit tutorial with examples: Setting up, writing, and running Java Unit tests 2022).” Mocking is used primarily with code that would have complex methods used to call databases, requiring objects that can be very large. With mocking, the object is separated from the code and used to test as necessary, which eliminates the need for creating the object many times.

As the requirements for the milestones did not contain a lot of object members, it was impractical to set up a mocking framework such as Mockito or PowerMock . A feature of JUNIT that was also not used was fixtures such as setup and teardown methods. Fixtures are used to ensure that tests can be repeated (Junit - Test Framework).

A technique not used in the milestones was also the use of JUNIT Test Suites and Test Runners, which are used to bundle unit tests and run them together (Junit - Test Framework). In a larger application with many files of object classes, a Test Suite and Test Runner would be helpful in order to provide more fluidity in the running of the test cases.

For each of the techniques you discussed, explain the practical uses and implications for different software development projects and situations.

For the technique of TestMethodOrder, DisplayName, and Order annotations, I found it very practical for the milestone projects as it provided a simple way to order and run the unit tests. For small projects such as the milestones or for quick checking in larger applications, I found these techniques to be very helpful in order to ensure that the tests would run in order as well as be able to read the details about test failures from the output screen in Eclipse. This is something that would also be helpful when testing a small part of a large application as a quick check to make sure that the developer is on track.

On application development teams that have a quality assurance engineer, the DisplayName annotation would be helpful to them to help ensure that the validation requirements are being tested. Some implications of only using the TestMethodOrder without a test runner or a test suite would be that it would be difficult to implement if the code base became much larger.

The naming convention used to name the test methods helps to describe what the purpose of the unit test is for, and this is helpful for any size project. Many times, there are unit tests that do not make sense and are hard to understand by additional developers or quality assurance engineers. A slight implication of the naming convention is naming a method with too many words, as every character in a unit test takes up storage in the program, and many times development teams are told to eliminate any extra words in order to alleviate that problem.

Code coverage is a technique that is practical in large and small software development projects, with many employers setting a code coverage percentage standard for application development. Code coverage is helpful to the developer to ensure that the functionality of their code is tested, yet an implication can be that the developer feels compelled to test functionality that is not necessary in order to provide a higher code coverage percentage. This implication can lead to a bloated code base, which makes the unit tests take longer to run and can affect the build time of a project if unit tests are a requirement of the build tool.

Assertions are a necessary part of JUNIT test development, but as with code coverage, assertions may also be overused. It is best to test small parts of code at a time and check on whether the assertion is covering the functionality of the code, rather than adding to the bloat and creating the same undesirable effect such as writing extra unit tests unnecessarily.

Mindset

Assess the mindset that you adopted working on this project. In acting as a software tester, to what extent did you employ caution? Why was it important to appreciate the complexity and interrelationships of the code you were testing? Provide specific examples to illustrate your claims.

An example of the extent that caution was employed was to the projects was to ensure that only the requirements were included in the code base in order for extra code or dead code to not be present. In order to appreciate the complexity and interrelationships of the code I was testing; it was necessary to only focus on what is required rather than add extra code that did not have a purpose. An example would be the object class for each project, which defines the necessary class variables and their respective getters and setters. In each object class, there are only the necessary variables in order to make the application fulfill the requirements.

A contact can be anything, but the requirement was for this service was that the Contact should have a String contactID, a String first name, a String last name, a String phone number, and a String address. If I added a String email address, for example, the code would be harder to test and contain requirements that do not exist. This is an example from the Contact object:

public class Contact {

private final String contactID;

private String firstName;

private String lastName;

private String Number;

private String Address;

private static AtomicLong idGenerator = new AtomicLong();

}

Assess the ways you tried to limit bias in your review of the code. On the software developer side, can you imagine that bias would be a concern if you were responsible for testing your own code? Provide specific examples to illustrate your claims.

To limit bias in the review of the project code, it was important to only work with the information that was required, such as not adding any extras as well as ensuring that any required validations were present and tested in the JUNIT tests.

I can imagine that bias is a concern when creating and testing your own code as developers have a timeline and want their code to work in order to get their job done. Developers may also find different ways to implement code and become argumentative when asked to defend their choice or to change their code by a business or quality engineer. When developing code, it is important for the developer to ensure that they keep the needs of the customer in mind and ensure that the requirements are met. It is important for the developer to ensure that their code can be tested and validated by others as a part of the software development process.

An example from the Appointment service project is that I would want outside testers to validate that I have included the proper validation requirement on the description variable:

// The description should not be null nor should it be greater than 50 characters

if (description == null || description.length() > 50) {

throw new IllegalArgumentException("Description cannot exceed 50 characters");

}else{

this.description = description;

}

I would ensure that the outside tester would be able to find that I have created a JUNIT test in order to test this functionality that was required as the code will throw an exception if the description is too long:

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

LocalDate date8 = LocalDate.now();

Appointment appointment5 = new Appointment(date8, "my second test appointment");

appointment5.setDescription("my second test appointment with a super long description for testing purposes");

});

Finally, evaluate the importance of being disciplined in your commitment to quality as a software engineering professional. Why is it important not to cut corners when it comes to writing or testing code? How do you plan to avoid technical debt as a practitioner in the field? Provide specific examples to illustrate your claims.

A developer is tasked with assisting a business or an individual with creating a software solution that will help them achieve their goals. It is important that a developer be committed to producing quality applications for many reasons. A quality application will be easy to test as well as easy to modularize with increased reusability. Software testing results are often kept as a permanent documentation record that will be utilized as a tangible deliverable in order to show that the requirements of the project have been met and the software is working as it should be.

As the software testing produces a tangible record, these records are important when there is rework asked for in the application as they prove that the original requirements were made and that the code wasn’t defective. Rework can be considered as tech debt is there is a piece of functionality that was not included in the software release, which will be added to a backlog as a miss by the development team. Defective code can cause many liabilities, such as high financial cost to a business, or even environmental damage and loss of lives, depending on the type of software that is written. It is important to have the proper software testing completed in order to show that there were no corners cut, that the business requirements have been fulfilled, and that the proper testing steps have been completed.

According to Martin (2022), software bugs can be traced back to as far as the 1960’s with the incident that happened to the Mariner Spacecraft in 1962. The Mariner was launched by NASA as an unmanned, data gathering space mission intending to fly past the planet Venus. The space craft had barely taken off when NASA was forced to issue a self-destruct command as it had veered off-course. The error was determined to be a software omission of a hyphen in a single line of code, resulting in an $18 million dollar cost ($169 million in today’s numbers).

Martin (2022) reiterates that software is “omnipresent and affects everything that we do.” Along with the monetary losses, there are also losses to a business’ reputation as well as potential service agreement violations. The negative effects extend to the privacy of the people involved, valuable data loss, and potentially the loss of lives (Martin, 2022).

References:

*Junit tutorial with examples: Setting up, writing, and running Java Unit tests*. Parasoft. (2022, April 20). Retrieved June 3, 2022, from <https://www.parasoft.com/blog/junit-tutorial-setting-up-writing-and-running-java-unit-tests/>

*Junit - Test Framework*. Tutorials Point. (n.d.). Retrieved June 3, 2022, from https://www.tutorialspoint.com/junit/junit\_test\_framework.htm

Martin, D. (2022, January 26). *11 of the most costly software errors in history*. Raygun Blog. Retrieved June 17, 2022, from https://raygun.com/blog/costly-software-errors-history/