CS320 Summary and Reflections Report

Kai I. Glass

Southern New Hampshire University

Table of Contents

[Table of Contents 2](#_Toc117338013)

[1. Summary 3](#_Toc117338014)

[1a. Describe your unit testing approach for each of the three features. 3](#_Toc117338015)

[1b. Describe your experience writing the JUnit tests. 4](#_Toc117338016)

[2. Reflection 4](#_Toc117338017)

[2a. Testing Techniques 4](#_Toc117338018)

[2b. Mindset 5](#_Toc117338019)

[References 6](#_Toc117338020)

CS320 Summary and Reflections Report

# 1. Summary

## 1a. Describe your unit testing approach for each of the three features.

### Contact Service.

#### To what extent was your approach aligned with the software requirements?

My testing approach aligned closely with the requirements. Some areas where I missed the requirements points is the update section where I created a new copy with different keys. I just finished fixing it to where only one copy is created, and all elements are updated at once. Another portion where I didn’t meet the requirements is the find by id section in which I didn’t implement any code. I also just fixed this portion, and all the code is running smoothly.Here is an excerpt of annotations I added to my contact entity,

@NotBlank(message = "ContactID is a required field")

@Size(max = 10, message = "ContactId cannot be longer than (max) characters)")

String contactID,

#### Defend the quality of your JUnit tests.

### . I know that my Junit test were successful because I also ran them in failing scenarios so I could better understand how the test work and that what I got as a success was what I wanted the code to do. I frequently tested my code coverage to ensure that I was meeting all requirements. When I first turned in my contact service assignment I hadn’t fully tested each variable in my contact class. I have since done so and my code meets all requirements.A screenshot of a computer Description automatically generated with medium confidence

### Task Service.

#### To what extent was your approach aligned with the software requirements?

My testing approach for the task service aligned with the software requirements. In the requirements the Task class should have an Id that cannot be empty, updatable, or longer than 10 charters. It should have a task name that cannot be empty, can be updatable, and cannot be longer than 20 characters. Finally, it should have a description that cannot be empty, can be updateable, and cannot be longer than 50 characters. As well as the Task class it should have Task service class which allows you to create a new task, update, find, or delete a task. Here is the code I used to ensure that my variables were tested following the requirements.

void checkTaskName(final String name) {

if(name == null || name.trim().length() < 1) {

throw new IllegalArgumentException("TaskName is a required argument");

}

}

void checkTaskDescription(final String description) {

if(description == null || description.trim().length() < 1) {

throw new IllegalArgumentException("TaskDescriptiom is a required argument");

}

}

#### Defend the quality of your JUnit tests.

I know my JUnit test were successful because I tested every branch. When I first turned in this assignment, I received the feedback that I needed to test the requirements for my task name and task description. I implemented test to make sure they were not empty or past the character limit. My test coverage exceeded 85%.A screenshot of a computer

Description automatically generated

### Appointment Service.

#### To what extent was your approach aligned with the software requirements?

My testing approach for the task service aligned with every software requirement. In the requirements section of the appointment service assignment, the appointment class should have an appointment Id that cannot be empty, updatable, or longer than 10 charters. It should have a appointment date that cannot be null, can be updatable, and cannot be in the past. Finally, it should have a description that cannot be empty, can be updateable, and cannot be longer than 50 characters. As well as the appointment class it should have appointment service class which allows you to create a new task, update, find, or delete a task. Test classes should also be implemented to ensure all code meets the requirements. Here is the code I used to ensure that my date was tested to not be in the past.

void checkAppointmentDate(final Date date) {

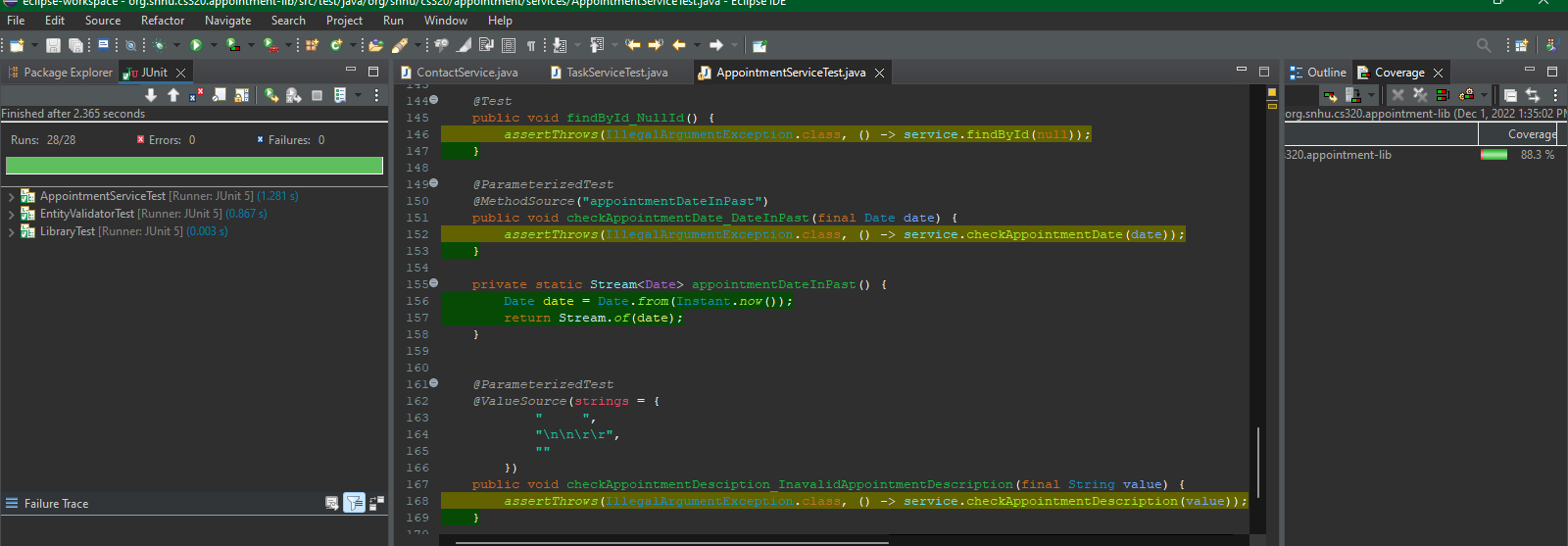
if(date.before(Date.from(Instant.now()))) {

throw new IllegalArgumentException("This date is in the past.");

}

}

#### Defend the quality of your JUnit tests.

My JUnit test was effective because I tested all possibilities including dates in the past. I communicated a lot with my professor on this assignment because I have never worked with the date class before. I also did a lot of research and was able to ensure that my date was tested by using a method source parameterized test.

## 1b. Describe your experience writing the JUnit tests.

### How did you ensure that your code was technically sound?

I ensured that my code was technically sound with the JUnit test by using assertions and annotations. I used java annotations in my public records to show the requirements of each of the assignments variables. I used assertions in all my test classes that would produce results that meet the requirements of the project. I also ensured that during coverage testing that every branch was met, and no statements were missed. I also ensured that no test failed prior to turning in the final deliverable.

### How did you ensure that your test code was efficient?

My code is efficient because I used parameterized tests where needed. To check the requirements of my variables I needed to preform parameterized tests for strings. Using these tests, I threw illegal argument exception because failures of these requirements should stop the code. Examples of this can be found in the very bottom portion of all of my service class and service test classes. I also used a version of parameterized test using a method source to ensure that my appointment date was not a date set in the past. This can be found in ApointmentServiceTest class lines 149-158. I also made sure my code was efficient by not testing against the pattern I set for the phone number because that would be anti-pattern testing.

# 2. Reflection

## 2a. Testing Techniques

### What were the software testing techniques that you employed in this project?\

In this class we have employed a variety of black-box and white-box testing techniques in our milestone. A black-box technique that I have used in every milestone is equivalence partitioning. The type of equivalence partitioning I have used has been input partitions. The way this works is that a program can accept certain inputs and reject others. In each milestone we were given requirements for variables for each of our classes that could accept certain inputs. I did equivalence partitioning tests to determine the values were not empty and under a max character limit. Another black-box technique we used was boundary value analysis. This is when you will accept a sequence of characters or integers but nothing outside of this boundary. We used this with the phone variable in the contact service with the pattern annotation to ensure that it was a valid phone number. A white-box technique that I implemented is the statement testing and coverage. This technique is the completion of the test is determined by whether the test executed the required statements. I used this in my milestones by using code coverage. Another white-box technique I used is decision testing and coverage. This technique is when you ensure the test is complete by executing all decisions in the test. I implemented this by ensuring all branches were met during code coverage portion of my project.

### What are the other software testing techniques that you did not use for this project?

In this project we also didn’t use a variety of techniques that could have made our project better. One example of this is the black-box technique called decision table. In this technique you make a table of all the input conditions and all the actions that can occur from them. Another type of black box testing I did not use is use case testing. This is when you create diagrams based on actors and the system to design the interaction that produce an output. The final method I did not use is state transition testing which is a black-box technique. This technique is when you test all different states of an object. A similar example of this would be all the different outputs you get when pressing mode on a watch.

### For each technique you discussed, explain their practical uses and implications for different software projects and situations.

For each of the techniques explained above there are practical uses and implication for different software development projects and situations. To begin I will start with black box testing techniques. For equivalence partitioning a practical use would be partitioning ages on a dating app, which also implies that this could be used for developers of those apps to ensure that the age range a person is looking for is the one they match up with. I will use the same example for boundary value analysis. For every dating app there is a distance function that makes matches near you or a designated radius. Many other apps use this function today and this testing can be used to implement it in development projects. The next technique is decision table testing. A very practical use for this is circuit design, in which they use function tables similar to a decision table to design a particular circuit. This can be used in development projects such as personality tests to determine how your responses would match you up with a specific personality. State transition testing is used in a lot of development projects such as watch apps where one can hit the track button an get multiple different outputs. Finally, use case testing is used in various different development projects. One I used use case testing on was an atm class that I created for a class a few terms ago were I created diagrams before I coded. The next portion we will discuss is white-box techniques. The first is statement testing and coverage, this should be used in every development project from phone apps to human resource training applications to determine every statement that you code is being tested. The same goes for decision testing and coverage. You want to make sure that you are testing a scenario for every decision that way the code doesn’t allow errors to get passed into the program.

## 2b. Mindset

### Assess the mindset you adopted working on this project.

I employed a lot of caution when working as software tester. I find it is important to appreciate the complexity and interrelationships of the code I am testing. During the code coverage portion of this project, I employed caution the most. When I first ran code coverage for my project it was very low. From there I went into every class and looked at the code and determined what needed to be tested. I realized that in order to test the requirements for each of my class variable I had to add code to the service class and then and multiple test in the service test classes. I also ran into a lot of trouble with the appointment date and using the date class. I did a lot of research and talking with my professor to ensure that I was testing the appointment date to the best of my ability and not hard coding dates in.

### Assess the ways you tried to limit bias in your review of the code.

On the software developer side of things, I can see where a bias can be formed if you were responsible for testing your own code. Confirmation bias is common when a developer tests their own code because it is hard to find one’s own mistakes. During this project I was a victim of this bias myself and had to reach out to my professor twice. It was during writing the appointment service milestone. The first problem that occurred was I didn’t know how to work with the date class. The second problem that occurred is that I didn’t know how to write a parameterized test to test that the date is not in the past. I found it best to combat this issue by taking a step away from the code for a few hours and come back when I have a different frame of mind and I can usually deduce where the issue is.

### Finally, evaluate the importance of being disciplined in your commitment to quality as a software engineering professional.

It is important not to cut corners when writing or testing your code because it can lead to higher costs and a poor product. If you cut corners in the testing phase, you can allow unwanted access to code that can end in hacking or disruption. If you cut corners in either phase it can result in more time to fix a problem. More time and resources equals more cost in the long run. The way I intend to avoid technical debt is by following all the testing techniques. I will use use case testing before I begin writing any of my code to ensure that what I am coding meets the requirements I have set out before me. I will use an agile framed method of sprints and frequent check ins to ensure that the product is moving in the right direction and there is not any major problems to address.

References

Hambling, B., Morgan, P., Samaroo, A., Thompson, G., & Williams, P. (2019). *Software testing :*

*An istqb-bcs certified tester foundation guide - 4th edition*. BCS Learning &

Development Limited.