CS320 Summary and Reflections Report

Max Gilhespy

SNHU

Table of Contents

Table of Contents …………………………………………………………………………....…… 2

1. Summary ………………………………………………………………..……………….. 3
2. Describe your unit testing approach for each of the three features ……..……………..… 3

i. To what extent was your approach aligned to the software requirements? ………. 3

ii. Defend the overall quality of your JUnit tests …………..………..…………….... 6

1. Describe your experience writing the Junit tests ………..…..………......................…… 12
2. How did you ensure that your code was technically sound? ................................ 13
3. How did you ensure that your code was efficient? …….………..…………...….. 13
4. Reflection .…………………………………………….………………………….…….. 14
5. Testing Techniques …………………………………………………………………..…. 14
6. What were the software testing techniques that you employed in this project? 14
7. What are the other software testing techniques that you did not use for this

project? …………………………………………………………………………. 15

1. For each of the techniques you discussed, explain the practical uses and implications for different software development projects and situations ……… 15

b. Mindset

1. Assess the mindset that you adopted working on this project ………………… 16
2. Assess the ways you tried to limit bias in your review of the code ….……...… 16
3. Evaluate the importance of being disciplined in your commitment to quality as a software engineering professional …………………………………………….. 16

**CS320 Summary and Reflections Report**

**Summary**

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

The software testing techniques employed for each of the three features were unit tests. Unit tests can use both static and dynamic testing. Static testing can be done manually by simply looking through the code but for larger projects it makes more sense to use automated tools that do the job. The software does not need to run for static testing. There was only a small amount of code needed for each feature so while looking through the code could have been enough to determine that they would function correctly, using Junit testing was a way to ensure that nothing was missed as could happen with only a manual review. As we do have to run the code to do JUnit testing that explains how JUnit tests are both static and dynamic.

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

Each class has specific requirements which were defined. The requirements for the Contact Class are specific String variables which are contactId, firstName, lastName, phone, and address. Further requirements are that none of the fields shall be null. Added to that the contactId field shall not be updatable. Other requirements are that contactId, firstName, and lastName must each be no longer than 10 characters, phone must be 10 digits in length and address must be no longer than 30 characters.

Here is an example of how firstName was set to align to the software requirements.

public void setFirstName(final String firstName) throws IllegalArgumentException {

// Check fistName meets requirements before setting

if (firstName == null || firstName.isBlank()) {

throw new IllegalArgumentException("firstName must have a value.");

} else if (firstName.length() > 10) {

throw new IllegalArgumentException("firstName cannot be longer than 10 characters.");

} else {

this.firstName = firstName;

}

}

The requirements for the ContactService Class are that contacts can be added, deleted, or updated and that this will happen based on the unique contact id in each contact object.

Here is an example of how firstName can be updated and shows that the contact id must be correct for the update to happen. This is consistent with the requirements.

public boolean updateFirstName(final String id, final String firstName) {

boolean updateComplete = false;

for(Contact contact : contacts) {

if (contact.getContactId() == id) {

contact.setFirstName(firstName);

updateComplete = true;

}

}

return updateComplete;

}

The requirements for the Task Class are specific String variables which are taskId, name, and description. Further requirements are that none of the fields shall be null. Added to that the taskId field shall not be updatable. Other requirements are that taskId must be no longer than 10 characters, name must be no longer than 20 characters, and description must be no longer than 50 characters.

Here is an example of how description was set to align to the software requirements.

public void setDescription(final String description) {

// Check description meets requirements before setting

if (description == null || description.isBlank()) {

throw new IllegalArgumentException("description must have a value.");

} else if (description.length() > 50) {

throw new IllegalArgumentException("description cannot be longer than 50 characters.");

} else {

this.description = description;

}

}

The requirements for the TaskService Class are that tasks can be added, deleted, or updated and that this will happen based on the unique task id in each task object.

Here is an example of how name can be deleted and shows that the contact id must be correct for the delete to happen. This is consistent with the requirements.

public boolean deleteTask(final String id) {

return this.tasks.removeIf(task -> (task.getTaskId() == id));

}

The requirements for the Appointment Class are specific String variables which are appointmentId and description. There is also a required Date variable. Further requirements are that none of the fields shall be null. Added to that the appointmentId field shall not be updatable. Other requirements are that appointmentId must be no longer than 10 characters and description must be no longer than 50 characters and description must be no longer than 50 characters. Finally, the date cannot be in the past.

Here is an example of how date was set to align to the software requirements.

public void setDate(Date date) {

// Check date meets requirements before setting

if (date == null) {

throw new IllegalArgumentException("date must have a value.");

} else if (date.before(new Date())) {

throw new IllegalArgumentException("date must must be in the future.");

} else {

this.date = date;

}

The requirements for the AppointmentService Class are that appointments can be added and deleted and that this will happen based on the unique appointment id in each appointment object.

Here is an example of how an appointment can be added and shows that a new appointment can only be added with a unique appointment Id. This is consistent with the requirements.

public boolean addAppointment(final Appointment newAppt) {

boolean apptExists = false;

for (Appointment appt : appointments) {

if (newAppt.getAppointmentId() == appt.getAppointmentId()) {

apptExists = true;

}

}

// If appointment doesn't exist, add it to the appointments list.

if (!apptExists) {

this.appointments.add(newAppt);

// Return true for adding appointment to list.

return true;

} else {

// Appointment already exists, return false for adding the appointment.

return false;

}

}

**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?**

For the Junit tests each method in each class was tested by using parameters that would work and then by using parameters that would not work. When all tests passed this meant that all requirements were met and all the specifics such as allowed lengths of variables and use of unique ids happened as expected. There were 187 unit tests and 187/187 unit tests passing with 88% code coverage.

Coverage verification for Grand Strand Systems Test Classes

AppointmentTest.java

A screenshot of a computer program

Description automatically generated

AppointmentServicesTest.java

A screenshot of a computer

Description automatically generated

ContactTest.java

A screenshot of a computer program

Description automatically generated

ContactServicesTest.java

A screenshot of a computer

Description automatically generated

TaskTest.java

A screenshot of a computer

Description automatically generated

TaskServiceTest.java

A screenshot of a computer program

Description automatically generated

**Describe your experience writing the Junit tests.**

My experience with writing the tests was to try and be sure that each part of each method had a test to ensure the coding would work as required. I used try-catch blocks to ensure that inputs were as expected. I also used try-catch blocks to ensure arguments were thrown when expected. I used assertAll blocks when more than one assert was needed. This was to make sure that if a test failed in a set it would continue with running the next test in that set.

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

In the ContactTest Class I have this code:

*@ParameterizedTest*

*@NullAndEmptySource*

*@ValueSource*(strings = {

" " })

public void testCreateContactWithInvalidNullOrEmptyId(String testId) {

String message = "(did not throw)";

try {

new Contact(testId, "fName", "lName", "0123456789", "address");

} catch(Throwable exception) {

message = exception.getMessage();

}

*assertEquals*("contactId must have a value.", message);

}

To ensure the code is technically sound I ran the test to see that it did what I was expecting it to do.

**How did you ensure that your code was efficient?**

The code from the ContactTest Class above is efficient as it is able to check for two things while only using one assert. It checks that certain conditions will throw an exception. It also checks that the exception throws the message that we are expecting to see that lets us know that the input is invalid.

Another way of keeping things efficient was that I used @ParameterizedTest in many places so that multiple parameters could be used with the same test and so I did not have to rewrite the same code for each different parameter multiple times. Using @NullAndEmptySource was another way to be efficient as it would add these parameters and so save time.

Here is an example from ContactServiceTest Class:

*@ParameterizedTest*

*@NullAndEmptySource*

*@ValueSource*(strings = {

" ",

"01234567890" })

void testUpdateFirstNameWithInvalidFirstName(String testFirstName) {

*assertThrows*(IllegalArgumentException.class, () -> testContactService.updateFirstName("2", testFirstName));

}

**Reflection**

**Testing Techniques**

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

The software testing techniques employed for each of the three features were unit tests. Unit tests are both static and dynamic testing. Static testing can be done manually by simply looking through the code but for larger projects it makes more sense to use automated tools that do the job. The software does not need to run to do static testing. There was only a small amount of code needed for the each class so while looking through the code could have been enough to determine that it would function correctly, using the Junit testing was a way to ensure that nothing was missed as could happen with only a manual review.

One characteristic of unit tests is that the software is tested in small pieces. The best outcome would be for every function to be tested. This would give 100% coverage of the class that is tested. Other characteristics are:

* The tests are repeatable.
* The tests will remain relevant into the future.
* The tests run quickly.

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

Integration testing was not used, and system testing was not used either. Acceptance testing was another technique that was not used. The black box testing technique was not used. Black box testing is the test of behavior as a program transitions from one state to another. Equivalence partitioning and boundary value analysis are black box techniques.

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

The practical use for unit testing is that it can be done from the beginning of creating code and is a way to know that each part is functioning as expected. Integration testing is necessary as the project becomes larger because it helps testers to detect problems that may have been more difficult to find if the software was allowed to grow without testing how each class and method work together. System testing is the testing at the end once the project is complete and the developers are looking for any final bugs. Acceptance testing is similar but should be the finished product once all bugs have been ironed out. The implication of equivalence partitioning is that the number of test cases can be reduced while still achieving comprehensive coverage. The implication of boundary analysis is that edge cases of inputs will not cause issues with the program.

The practical uses and implications of these software testing techniques will vary depending on the specific software development project. However, these techniques are a great way to improve the quality of the software created and should prevent errors from reaching the end user.

**Mindset**

**Assess the mindset that you adopted working on this project.**

My mindset was to be aware of the requirements and make sure that the code covered all aspects of what was asked.

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

I feel I did not have any bias when reviewing my code as I have been open to editing anything that could be coded in a more robust or efficient way. The beginning of coding is more like a draft than a final piece where you realize as you add more pieces in later it may mean finding that earlier pieces can be written in a better way.

**Evaluate the importance of being disciplined in your commitment to quality as a software engineering professional.**

It is very important to be disciplined and not cut corners when writing or testing code. There can be many unintended results from code failing which can include the risk on people’s lives. This kind of technical debt can be avoided if developers concentrate on coding to cover every requirement. Testing is vital and must be done vigorously to ensure that the finished program will run as expected without error.

References

GeeksforGeeks. (2022, March 27). *Software testing - boundary value analysis*. GeeksforGeeks. https://www.geeksforgeeks.org/software-testing-boundary-value-analysis/