Aneudi Matos

Linda Gaston

CS – 320

13 August 2024

Project Two

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

The tests for the Tasks module of the software were directly aligned with the software requirements. For example, one of the software requirements for the Task object was that the taskId cannot be longer than 10 characters, cannot be null and cannot be updated. To test written to test the character length was.

@Test(expected = IllegalArgumentException.class)

  public void taskIdTooManyCharacters() {

    Task task = new Task("1234567891111", "clean windows", "Clean the windows and spray with rainx solution");

  }

The test was to expect to return a IllegalArgumetnException returned by the task class because the id set by the test was > 10 characters. The test returned the expected output and passed. As for the updatable portion of the requirements a set method for the TaskId was left out to make the id immutable and not updatable after creation. The tests were aligned directly to the software requirements because it being tested to see if it meets those requirements.

**Defend the overall quality of your JUnit tests for the contact service and task service. In other words, how do you know that your JUnit tests were effective on the basis of coverage percentage?**

The contact service and task service had similar requirements and the coverage for both is one I am confident in. The reason for the confidence is that I carefully read the prompt and what the requirements for the software were, what needed to be updatable, what could not be null, the length of each string and how it would be deleted.

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

One of the sections in which the code was efficient was when the requirements for the taskId, taskName and description were being implemented.

        if (taskId == null || taskId.length() > 10) {

            throw new IllegalArgumentException("Invalid task ID ");

        }

Here the null and length of taskId are both being screened on the same line and both will return the IllegalArgumentException if true.

**What were the software testing techniques that you employed for each of the milestones? Describe their characteristics using specific details.**

For each of the milestones I used unit testing and each individual method was tested individually, without relying on the output from the other methods or tests to ensure good coverage and success. Error handling was used in the setter method for the date.

public void setDate(Date date) {

    if (date == null || date.before(new Date())) {

      throw new IllegalArgumentException("Invalid date, choose one in the future");

    }

    this.date = date;

   }

This ensures that when the test or code was run that the invalid date was not set and instead an error was passed. This was also used in the other milestones but not in the setter methods themselves.

Integration testing was used in the AppointmentService class by testing how the different components work together to make a new appointment.

  public void addAppointment(String appointmentId, Date date, String description) {

    String id = Integer.toString(currentId);

    Appointment newAppointment = new Appointment(id, date, description);

    ApptList.add(newAppointment);

    currentId ++;

  }

The Appointment service adds new appointments using the methods from the appointment class. Testing the integration and if the classes and methods are working together seamlessly.

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

Since this is a small project with no UI performance, testing was not necessary. Normally if it was software that was going to be delivered to a customer you would want to make sure that it could handle stress at different loads. Also check its response time to make sure output is being delivered efficiently. Security testing was also something that was not done on this project, this would be to make sure the private methods were safe and things that weren’t meant to be changed or updated were not accessible.

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

For error testing and handling the practical use is to make sure software is reliable and consistent, this makes sure the software is delivering accurate results. Implications are almost none but if small mistakes can cause the software to not return the desired output since the software is following a strict set of instructions. Integration testing is vital for testing complex workflows to make sure everything is communicating and working together as intended. The implications of this type of testing is if some of the software is “off the shelf” from another company which might make it difficult to do integration testing, because now there is two development teams involved.

**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.**

Testing with test cases was new to me so this required me to think differently than before, I was required to more deeply understand the dependencies and interactions. The complexity of the interactions is important to understand because you want to make sure your test can correctly test each layer, to ensure test accuracy.

**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 eliminate bias, always adhere to the standards and check for proper error handling. Also, when testing your own code don’t be overconfident and test as thoroughly as possible. When possible, have your test cases be peer reviewed.

**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.**

The commitment to quality as a software professional is of utmost importance, and one must not cut corners because like my peers and I discussed in module seven people can end up severely injured or dead. If your code was the one running on the Therac – 25 radiotherapy machine and because you tested the software extensively and added error handling, those six people could have been saved. To avoid technical debt, make sure the software is extensively tested, and your coverage is at least eighty percent, also go over you diagrams and notes to make sure everything that was implemented is being tested correctly and the right output is being returned.