Jason Yaeger

Project Two

12/11/22

I organized this within the outline for clarity and used some of the work from the milestones and prior modules.

1. **Summary**
   1. Describe your unit testing approach for each of the three features.
      1. To what extent was your approach **aligned to the software requirements**? Support your claims with specific evidence.

My approach was directly aligned with the software requirements as it was clearly outlined in the rubric what needed to be done. Some specific evidence for this can be seen in the input validation within the code that is reflected by the following criteria being met in task for example.

“The task object shall have a required unique task ID String that cannot be longer than 10 characters. The task ID shall not be null and shall not be updatable.

The task object shall have a required name String field that cannot be longer than 20 characters. The name field shall not be null.

The task object shall have a required description String field that cannot be longer than 50 characters. The description field shall not be null.”

* + 1. 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?

My JUnit tests were effective in assessing the quality of the code itself. This can be seen by looking at the coverage percentage of the various files. Most of the files are over 90% coverage. It seems quite difficult to reach very high percentages and I would imagine it is rare to get 100%.

* 1. Describe your experience writing the JUnit tests.
     1. How did you ensure that your code was **technically sound**? Cite specific lines of code from your tests to illustrate.

To ensure that JUnit tests codes were technically sound, I made sure to not only test it after it was written, but to think about what I need the code to do before writing anything. Planning and preparing for what the test needs to cover, makes it so that attention can then be placed on the details and things aren't missed. Some specific lines of code to illustrate this would be the following code to test if the delete contact function works correctly,

@Test

public void testRemove()

{

ContactService cs = new ContactService();

Contact contact = new Contact("1413252", "Jane", "Doe", "4444444444", "Sample 24 Drive");

cs.addContact(contact);

assertTrue(cs.deleteContact("1413252"));

//assertEquals(true, cs.addContact(contact));

}

* + 1. How did you ensure that your code was **efficient**? Cite specific lines of code from your tests to illustrate.

Efficient code is effective and function oriented. There are no unnecessary lines or unneeded wordiness to it. My favorite examples of this are always in doing the setters and getters. Some lines of code to illustrate can be seen below,

public Contact(String contactId,String fName,String lName,String phone,String address){

if(contactId.length() <= 10 && contactId != null) {

this.contactID = contactId;

}

this.setFirstName(fName);

this.setLastName(lName);

this.setPhoneNumber(phone);

this.setAddress(address);

}

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

One of the more interesting parts of the current module five was utilizing the date functions and various testing done with dates. This seemed to be tricky at first but the note in the module that said to use the before function to compare dates made it simpler. I then just created a date object set to the current date and was able to compare it to the appointment date. I did if statements with throws for validation; try statements with catches and assertequals in testing. These things would fall under static and dynamic testing like unit tests and functional tests.

* + 1. What are the **other software testing techniques** that you did not use for this project? Describe their characteristics using specific details.

Some techniques I did not use for the milestone would be black box testing, and automated testing such as what would be done through cucumber or katalon. The concept of black box testing is interesting as it almost puts extra rigor on the software because the person testing has no idea as to the design of it and they are using the software as an end user would. Automated testing I also had to read and learn more about, it seems useful however most people dislike katalon a lot and are recommending selenium. This seems to be done mostly by quality assurance testers/engineers.

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

Every software project will have its unique challenges and problems to solve. Practical uses for the previously discussed techniques would be say a webapp needs testing done they could have someone from a different department like marketing come in and do some black box testing for the development/testing team. Perhaps they could also use software like selenium to perform automated testing as well. Static testing and dynamic testing will nearly always be done by the developer while they are working and writing code, there does not need to be a dedicated tester or separate team in many cases.

* 1. Mindset
     1. 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.

I adopted a methodical mindset throughout this course and often found myself spending more time thinking and planning. Once the actual writing of code occurs there will be few questions left to be answered. I employed caution when designing the tests and code, and also then writing it. It is important to appreciate the complexity and interrelationships of the code as the coverage of unit tests and quality of code depended on it for most modules. TaskService depended on Task, etc. Operating withing packages and using assets from other classes or files was common. An example to illustrate can be seen here,

service.addTask(new Task("0000000001", "Singing", "Wedding Engagement"));

* + 1. 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.

The best way to limit bias I think is unrelated to anything technical or domain specific and more just a reflection of one's character and mental state in general. If a person is confident and secure in themself, they will have no issue acknowledging mistakes, learning from errors, and being self-critical in a healthy way. Limiting bias is about being in a good state of mind when working, going for walks, eating healthy, things like this are ways to limit bias that might not seem obvious. For these reasons I cannot imagine bias being a big concern for me personally right now, but I can understand how it is a potential issue in general. An example in a corporate environment, however, I would imagine it is safer and better policy to have multiple eyes and teams looking at code. It is more careful/through to have many people reviewing code as it is easier to overlook errors at times as one individual.

* + 1. 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.

With how competitive software is, and the businesses that rely on them, it is imperative for the success of software that the engineer has discipline and a commitment to quality. It only takes one bad experience for a user to move to a competitor or write off an application or piece of software entirely. This can be seen across industries. It is also why I find myself so interested in UX design as often the creative problem solving involved in designing to balance business objectives and advocating for the consumer is fascinating and rewarding. Cutting corners can lead to errors and huge costly problems so it is best to be thorough. I plan to avoid technical debt by being flexible and doing lots of planning, while developing iteratively and getting lots of feedback along the way. Examples of great development can be seen in products and companies that empower users and truly listen to their feedback, while using all the time and resources available to maximize and align these goals. Discord comes to mind as a great product fitting of this description.