project two submission

CS – 320 Software Test, Automation, and Quality Assurance

Southern New Hampshire University

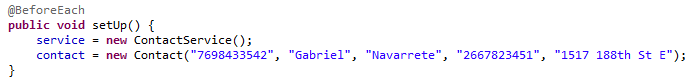
Andrea Carmen Sherry

1. **Summary**
   1. **Describe your unit testing approach for each of the three features.**During Project One, I developed and tested three core features, these being contact, task, and appointment services. Each of these features required a thorough approach to unit testing to ensure its functionality, reliability, and compliance with customer requirements.
      1. **To what extent was your approach aligned to the software requirements? Support your claims with specific evidence.**For “Contact Service” I used a combination of boundary testing, equivalence partitioning, and exception handling tests. Boundary testing ensured that input values at the limits of acceptable ranges worked correctly. Equivalence partitioning confirmed that different inputs producing the same output were handled consistently. Exception handling tests confirmed that invalid inputs were properly managed. This approach aligned with the requirements for strong input validation and error handling. For example, ensuring that the contact service handled correctly invalid email formats was a specific requirement. This can be seen in the following code from ContactServiceTest.java:   
         **A computer code with many text

         Description automatically generated with medium confidence**  
         The testing strategy used for the task service was focused on functional testing, ensuring that every feature worked as expected. I also used state transition testing to ensure that every task transitioned properly through different states. These functional tests were important to meet requirements related to task management operations. State transition tests ensured compliance with business rules for task status changes. An example of code from my TaskServiceTest.java is the following:   
         **A computer screen shot of a program

         Description automatically generated**  
         Finally, for the appointment service I implemented a scenario-based testing and a time-related testing. The way this works is that the scenario-based tests simulate those patterns found in the real-world usage and the time-related test verifies that the service handles correctly appointments across the different time zones and changes like daylight saving. An example of code from my AppointmentServiceTest.java is the following:   
         **A screenshot of a computer

         Description automatically generated**
      2. **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 quality of my JUnit tests was high, with over 80% code coverage, showing that the tests completely exercised the codebase. The high coverage percentage shows that the tests covered most of the code paths, including edges and exception handling.
   2. **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 my code was technically sound I followed best practices and used meaningful test names that made sense based on what our client requested and ensured to set up complete test data. For example, for “ContactServiceTest”, I used the following:   
         **A computer screen shot of a computer code

         Description automatically generated**  
         In this picture, the method addContact ensures that a contact is properly added to the list.
      2. **How did you ensure that your code was efficient? Cite specific lines of code from your tests to illustrate.**I ensured that my code was efficient by reducing unnecessary setup code. Just like on the following screenshot of my code:  
         ****  
         Using @BeforeEach helped me initialize test data once for every method.
2. **Reflection**
   1. **Testing Techniques**
      1. **What were the software testing techniques that you employed in this project? Describe their characteristics using specific details.**For this project I used boundary testing that helped me test edge cases to ensure that input limits were properly handled. Next, I used exception handling tests that helped me verify that the system was properly handling invalid inputs like duplicate IDs or null values. Last, equivalence partitioning helped me by grouping inputs that should be treated the same and tested representative values.
      2. **What are the other software testing techniques that you did not use for this project? Describe their characteristics using specific details.**One software testing technique that I did not use for this project is performance testing. This type of testing focuses on the system performance under load, which was not within the scope of unit testing. Since this technique is used in later stages of testing to ensure that the application can handle high volumes of data, I did not have to use this technique. Another technique not used is integration testing. This type of testing ensures that different modules work together properly. The reason why I did not use it is because it was beyond the scope of unit testing as well. A way in which I would have used otherwise this testing technique is to verify that the different services worked correctly with each other.
      3. **For each of the techniques you discussed, explain the practical uses and implications for different software development projects and situations.**Boundary testing is important in applications needing precise input validation, like financial systems, where it is used to validate transaction limits. By testing the edge cases, developers ensure that the application controls minimum and maximum values correctly, preventing errors at the boundaries of input ranges. Effective boundary testing improves the toughness and reliability of the application by ensuring that all edge cases are handled appropriately, preventing unexpected behavior and potential crashes, and leading to a more stable and user-friendly application. Equivalence partitioning is used to lower the number of test cases needed while covering a wide range of input scenarios. This technique is particularly effective in applications with large input spaces, like form validation in web applications, where inputs can be grouped into equivalent classes that should be treated the same. By ensuring complete testing with fewer test cases, equivalence partitioning makes the testing process more efficient and effective, helping to identify defects early in the development cycle, leading to higher quality software and reduced testing time and costs. Exception handling tests are important for applications that need to handle unexpected inputs properly. This is important in any system that interacts with users or other systems, where strong input validation and error handling are essential to keeping application stability and user satisfaction. Strong exception handling improves the application's stability and user experience by preventing crashes and providing meaningful error messages. This helps build trust with users and ensures that the application can handle real-world scenarios without failing. Integration testing is vital for verifying that different modules work together seamlessly. This is important in complex applications with multiple components, like those using a microservices architecture, where the interaction between services needs to be tested completely. Proper integration testing helps identify issues in the interaction between components early, reducing the risk of failures in a live environment. It ensures that the integrated system meets all functional requirements and behaves as expected, providing confidence that the system works. This leads to higher quality software and a smoother deployment process.
   2. **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.**Adopting a cautious mindset was essential as it helped me anticipate potential issues, as well as to handle code interdependencies. For example, while testing the “AppointmentService”, I made sure that creating overlapping appointments showed the proper error. By using this approach, I was able to identify difficult scenarios that could somehow break the application.
      2. **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 biases I made sure I reviewed my code from a tester’s perspective, ensuring that I looked for any flaws in it. For example, I questioned my assumptions about valid inputs and wrote tests to challenge these assumptions. An example is when I involved testing date formats to ensure that “AppointmentService” handled valid and invalid formats properly. By doing this, I was able to create a strong and reliable application.
      3. **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.**It is important not to cut corners when it comes to writing or testing code because it can lead to have hidden bugs and technical debt. By sticking to thorough testing, I ensured code quality. For example, instead of assuming that the time zone handling in the “AppointmentService” was correct, I wrote detailed tests to confirm it. Additionally, I plan to avoid technical debt by planning regular code reviews and refactoring sessions that will help me manage technical debt. To detect any issues early in the process, I will continue to use automated testing frameworks and maintain high test coverage.

**References**

:                                                  Thomas Hamilton, & Hamilton, T. (2024, March 19). *Boundary Value Analysis and equivalence partitioning*. Guru99. https://www.guru99.com/equivalence-partitioning-boundary-value-analysis.html

*Integration testing*. Tutorialspoint. (n.d.). https://www.tutorialspoint.com/software\_testing\_dictionary/integration\_testing.htm

Sherry, A. C. (2024). Contact, Task, and Appointment Services Project. Unpublished project.