# Project Two

# Andrew Piccirilli

# SNHU CS-320

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.

if (phoneNumber == null) {

throw new IllegalArgumentException("Phone number requires input");

} else if (phoneNumber.length() != ***PhoneNumLength***) {

throw new IllegalArgumentException("Invalid Phone Number Length. Number must be " +

***PhoneNumLength*** + " digits.");

} else if (!phoneNumber.matches(regex)) {

throw new IllegalArgumentException("Invalid Input, Phone Number must contain only numbers");

} else {

this.phoneNumber = phoneNumber;

* + 1. In the contact class this snippet of code shows testing to verify that the phone number has input, is within valid length, and doesn’t contain characters other than numbers. This lines up with the requirements set forth for the contact service classes where the phone number had to meet those requirements. Similar testing was done on the other aspects in Contact classes in order to make sure that the requirements were met.
    2. void validateDescriptionLength() { in the tasktest file tests to make sure that the description length isn’t longer than the required maximum length for descriptions as described in the task classes requirements.
    3. In the Appointment service classes this test in appointmentServiceTest tests to ensure no duplicate appointments as required in the system requirements.

1. void testAddingDuplicateAppointmentThrowsException() {
2. appointmentService.addAppointment(validAppointment);
3. Appointment duplicateAppointment = new Appointment("12345", new Date(System.*currentTimeMillis*() + 10000), "Teeth Cleaning");
4. *assertThrows*(IllegalArgumentException.class, () -> appointmentService.addAppointment(duplicateAppointment));
5. }
   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?
      1. Utilizing assertThrows to verify error management and assertAll for collective assertion validation enhanced the thoroughness of functionality and constraint testing. This suggests extensive test coverage, adeptly capturing a wide array of possible outcomes and issues.
6. 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.
      1. The use of assertThrows() to verify that specific actions lead to expected exceptions, such as in testSettingAppointmentDateInPastThrowsException(), ensures that the system behaves correctly under invalid conditions.
   2. Efficiency: By designing tests that reuse setup components (@BeforeEach) and directly target the intended functionality, the code avoids unnecessary repetitions and focuses on validating key behaviors efficiently.
7. **Reflection**
   1. Testing Techniques
      1. What were the **software testing techniques** that you employed in this project? Describe their characteristics using specific details.
         1. Black-box testing is used in the newContactTest() method of ContactServiceTest, where the focus is on the input and output of the newContact method without regard to its internal workings. Boundary testing is shown in tests that input just at or beyond the limits of valid data lengths.
         2. White-box testing, which is evident from the direct testing of internal logic and validation mechanisms within both the Appointment and AppointmentService classes. This testing involves knowledge of the internal structures to design test cases
            1. testSettingInvalidDescriptionsThrowsException relies on understanding how the Appointment class processes and reacts to invalid input, specifically testing the behavior of its internal error handling.
      2. What are the **other software testing techniques** that you did not use for this project? Describe their characteristics using specific details.
         1. There were multiple testing techniques that weren’t used such as
            1. Performance Testing which evaluates how the system performs under a particular workload, focusing on responsiveness, stability, and resource usage. It is crucial for applications requiring high efficiency and reliability under various conditions. For platforms that demand superior performance, such as web servers with high user traffic or software analyzing large data, performance testing is crucial. It aids in pinpointing system inefficiencies and enhancing both the speed and capacity of the application, ensuring it can handle expected demands efficiently.
            2. Usability Testing which assesses how user-friendly the application is, focusing on the user's ease of using the application and the intuitiveness of the UI. It's more common in applications with graphical user interfaces. Useful for customer-facing applications, websites, and software with GUIs. It ensures that the end-users can easily navigate, understand, and use the software, leading to a better user experience and higher satisfaction.
            3. Security Testing that involves identifying vulnerabilities, threats, and risks in the software to prevent malicious attacks. This type of testing is vital for applications handling sensitive data or requiring authentication. For applications dealing with sensitive information, e-commerce platforms, and any system accessible over the internet.
   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.
         1. I had to do a lot of research/going back over material in order to implement the processes/test which I used in this project. Some examples of caution, in the Task class, the method validateInputs(String input, int maxLength, String errorMsg) shows a cautious approach by enforcing constraints on input values, thereby appreciating the complexity and ensuring robustness (if (input == null || input.length() > maxLength) { throw new IllegalArgumentException(errorMsg); }). This was to prevent future issues by addressing potential input-related errors upfront.
      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.
         1. Bias limitation is shown in my approach to testing, such as in TaskTest and TaskServiceTest, where scenarios cover a wide range of inputs and operations. Testing for null inputs and validating string lengths (Assertions.assertThrows(IllegalArgumentException.class, () -> { new Task(null, "Clean Room", "Make bed, pick up laundry, fold clothes"); })) illustrates my effort to objectively evaluate the code beyond developer assumptions.
      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.
         1. Through consistent and thorough testing, the purpose of this project is to demonstrate a proactive approach towards maintaining and enhancing the software's usability and robustness. Through the use of detailed testing I attempt to avoid the gradual buildup of unresolved problems that could impact the system's efficiency and reliability.