Summary Reflection and Reports

Throughout the course, specifically in module 3, module 4, and module 5, I was able to utilize Junit testing method to fulfill the software requirement checklist for every module. Junit is a framework of Java Object Oriented Programing in a test-driven development environment. By using that method, I can be able to double-check the functionality of the code whether it’s good or fail within the verification category. I think my approach is aligned with the software requirements because I was able to demonstrate the testing process clearly for each classes and methods on the basis of Junit.

In general, Junit was applied to verify the null and character limitation on String variables for every classes including Contact, Task, and Appointment. Specifically, module 3 was containing the majority of verification tests since the list of requirements is more targeted the updated String variable for firstName, lastName, phoneNumber, and address in ContactService.java. In addition to that, we also have to verify the updated name, and description in TaskService.java. Furthermore, Appointment.java introduce us a brand new Date object in the class attribute, compared to the all the String variables we have in Contact.java and Task.java.

As I have mentioned the requirement check list, since all the class using the same pattern of requirement on null and character limitation condition with String variable, we also have to generate unique ID for every Contact, Task, Appointment. For the method of services, the function should be able to add or delete and update per unique ID. For example, Task class have String taskName that cannot be null, cannot be longer than 20 chars, and taskName will be updatable, if else statement was applied to set the condition for that in Task.java

*public void setTaskName(String taskName) {*

*if (taskName == null) {*

*throw new IllegalArgumentException("Task name cannot be null");*

*} else if (taskName.length() > 20) {*

*throw new IllegalArgumentException("Task name cannot be longer than 20 chars");*

*}*

*this.taskName = taskName;*

*}*

In TaskService.java, we can use Boolean to set the condition true/false in order to verify whether the new taskName is identical with the previous one. If that’s not the case, this is how the condition is set up:

*public boolean updateTaskName(String taskID, String taskName) {*

*Task task = taskList.get(taskID);*

*//task.setTaskName(taskName);*

*if (taskName == task.getTaskName()) {*

*System.out.println("Name already exist!");*

*} else if (taskName.compareTo(task.getTaskName()) < 0 || taskName.compareTo(task.getTaskName()) > 0) {*

*task.setTaskName(taskName);*

*return true;*

*}*

*return false;*

*}*

As a result, Junit testing for both Task.java and TaskService.java will cover the functionality of these methods in order to give out pass or fail result.

*@Test*

*@DisplayName("Task ID is null will throw errors")*

*void testTaskID\_case2() {*

*Assertions.assertThrows(IllegalArgumentException.class, () -> {*

*new Task(null, "Task Name","Task Description");*

*});*

*}*

*@Test*

*@DisplayName("Task Name is longer than 20 chars will throw errors")*

*void testTaskName\_case1() {*

*Assertions.assertThrows(IllegalArgumentException.class, () -> {*

*new Task("TaskID1234", "Task Name More Than 20 Chars","Task Description");*

*});*

*}*

*@DisplayName("Should be able to add task with unique ID")*

*void shouldAddTask() {*

*taskService.addTask(taskID, "Task Name", "Task Description");*

*assertFalse(taskService.getAllTask().isEmpty());*

*assertEquals(1, taskService.getAllTask().size());*

*}*

*@Test*

*@DisplayName("Should be able to delete tasks per taskID")*

*void shoudDeleteTask() {*

*taskService.delete(Task.getTaskID());*

*assertFalse(taskService.taskList.containsKey(Task.getTaskID()));*

*}*

*@Test*

*@DisplayName("Should be able to update name")*

*void shoudUpdateTaskName() {*

*taskService.addTask(taskID, "Old Task Name", "Test Desc");*

*assertTrue(taskService.updateTaskName(Task.getTaskID(), "New Task Name"));*

*}*

These were the Junit testing that was employed in module 3, 4, and 5. The test cover majority of the requirements. Another testing technique could be incorporate with Junit to increase the test coverage more effectively would be Mockito, which is another framework for Java object-oriented programming.

Caution should be the mindset for every tester during the software testing. When it’s too good to be true, a good tester should consider going back to the requirement list to verify whether he/she understand the condition clearly in order to write out the pass/fail result on the verification test. For example, the updated method for Contact.java, Task.java, require the service to get the unique ID for every updated String in order to proceed with the update accordingly. If one was not cautious enough, they could be easily deviated from the original requirement; hence, the test coverage would be reduced significantly.

Bias should be avoided completely because it can hinder a good result. Also, it could be a concern if one were responsible for testing because bias can replace a good outcome with a bad one, assuming that the tester believes there was nothing wrong with their test structure and keep testing their faulty code.

Beside bias while writing the test code, discipline is an another important characteristic that one should employ in their mindset; in order to become a quality software engineers. In general, cutting corner should be throw away entirely since it could cause the instability in the code structure. Consistency is the key to open the door to a good code structure since it makes the code clean. My initial plan to avoid technical debt is creating a cleaner code with clear variables, intelligible functions, so it can be easy to recognize, understand and maintain. By writing a little a bit of code, testing it, then proceed to do the next, one by one, I was able to finish the project one in a timely manner.

References

Mariani, R. (2020, November 19). *Unit Tests: Power and Caution - The Startup - Medium*. Medium; The Startup. https://medium.com/swlh/unit-tests-power-and-caution-191e72f47afd

‌

*How to Write Test Cases in Java Application using Mockito and Junit*. (2022, February). GeeksforGeeks; GeeksforGeeks. https://www.geeksforgeeks.org/how-to-write-test-cases-in-java-application-using-mockito-and-junit/#

‌

Koopmans, T. (2019, March 13). *How cognitive biases influence software development*. Work Life by Atlassian; Atlassian. https://www.atlassian.com/blog/add-ons/how-cognitive-biases-influence-software-development

‌