**Summary and Reflections Report**

Abdullah Arif Jafri

Southern New Hampshire University

CS 320

Professor Omar Toledo

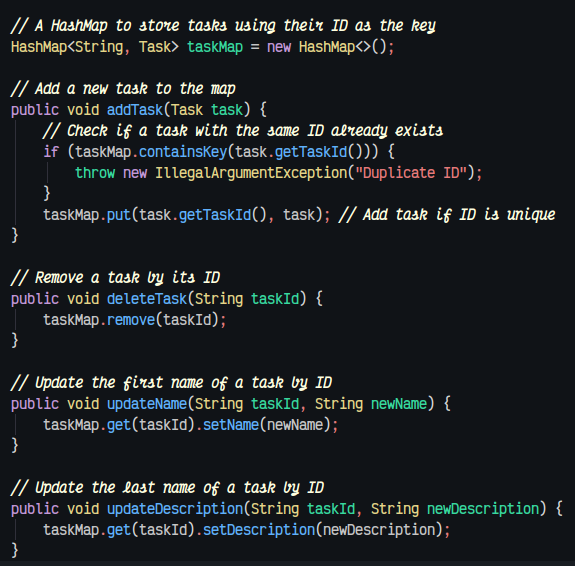
October 18th, 2025

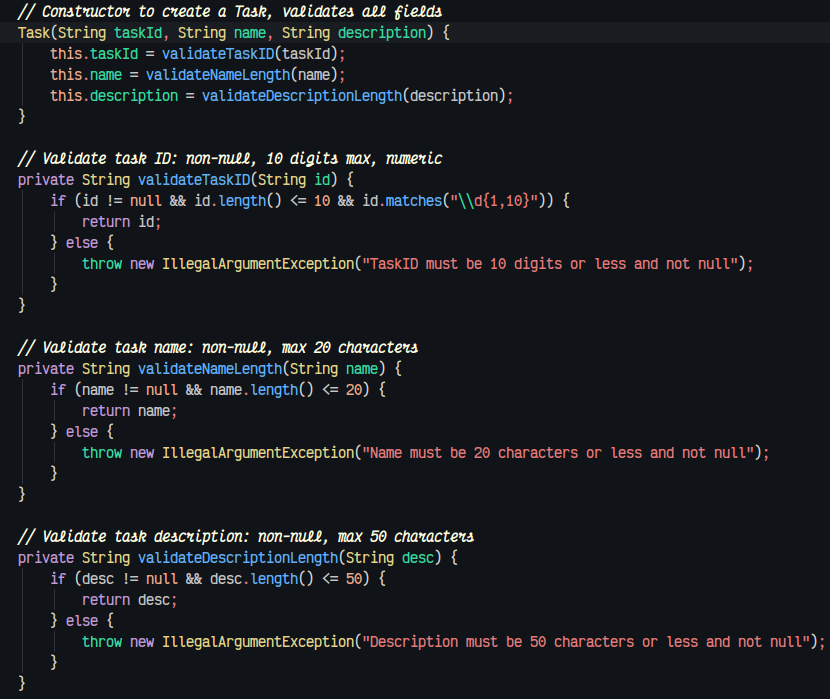
**Summary**

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 testing approach was heavily aligned with the software requirements for all the three testing. I was verifying the invariants in the software requirements description heavily and making absolutely sure I was following the requirements to the dot .

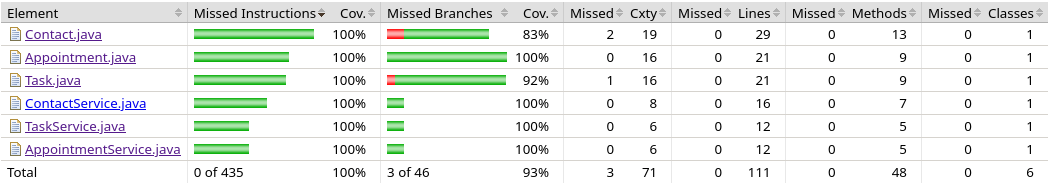
As an example let's take the task creation task. In that I focused on testing three main features: task creation, task updates, and task management through the TaskService class. For task creation, I verified that the constructor correctly enforced input constraints such as taskId being numeric and no more than 10 digits, name being under 20 characters, and description being under 50 characters. For task updates, I tested both valid and invalid setter operations to confirm that input validation was consistently applied. Finally, for task management, I tested adding, deleting, and updating tasks using TaskService to ensure the service behaved correctly when handling duplicate IDs or non-existent tasks.





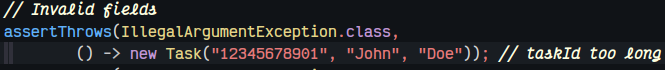
* 1. **Defend the overall quality of your JUnit tests. In other words, how do you know your JUnit tests were effective based on the coverage percentage?**

I am sure that my JUnit tests accurately cover all the possible edge cases because my Jacoco test reports indicated a 93% test coverage. This confirmed that all constructors, getters, setters, and service methods, including exception-handling branches, were thoroughly tested and provided evidence that my tests effectively verified the intended functionality and robustness of the system.



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.**

I ensured technical soundness by explicitly validating both inputs and outputs in my tests and verifying that exceptions were thrown for invalid cases. For example, I tested constructors to reject non-numeric or overly long id’s.



I also tested setters and service updates for boundary violations and null-safety. By covering both valid and invalid scenarios, I confirmed that my code behaves correctly under all expected technical conditions and enforces the constraints defined in the system requirements.

* 1. **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 using a HashMap in the TaskService class to store tasks instead of an ArrayList.

Eg - HashMap<String, Task> taskMap = new HashMap<>()

This allows constant-time operations for adding, deleting, and updating tasks. For example, in the addTask method, I check for duplicates and then put the task in the map. taskMap.put(task.getTaskId(), task) . Similarly, deleting a task uses taskMap.remove(taskId) and updating uses taskMap.get(taskId).setName(newName) or setDescription(newDescription).

**Reflection**

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

We employed several software testing techniques to ensure the quality and correctness of our work. One of the primary techniques was unit testing, which involved testing individual components or units of code in isolation to verify that they functioned correctly. In addition, we conducted functional testing, which focused on evaluating the software’s behavior against the specified requirements. This ensured that the implemented features met the desired functionality and aligned with the project’s objectives.

* 1. **What are the other software testing techniques that you did not use for the milestones? Describe their characteristics using specific details and for each of the techniques you discussed, explain the practical uses and implications for different software development projects and situations.**

There are several software testing techniques that we did not use for the milestones, each serving a specific purpose in ensuring software quality.

**Performance Testing -** Performance testing evaluates the software’s responsiveness, stability, and resource utilization under various workload conditions. This type of testing is particularly useful in projects where high user demand or heavy data processing is expected, as it helps ensure the software can handle peak loads without crashing or slowing down.

**Security Testing -** security testing focuses on identifying vulnerabilities, weaknesses, or potential exploits in the software. By assessing risks such as unauthorized access, data breaches, or integrity issues, security testing protects both the software and its users. This is critical in applications that handle sensitive data, such as financial systems, healthcare platforms, or any service involving personal information.

**Regression Testing -**  Regression testing is a technique that ensures new code changes do not negatively affect existing functionality. It is particularly valuable in ongoing software development projects, where frequent updates or feature additions are made. Regression testing helps maintain software stability and prevents unintended side effects from being introduced during development.