# **MILESTONE 2** -- SFT221 SCRUM Report and Reflections

This report should be completed in the class and submitted at the end of class. Late submissions cannot be accepted without prior approval of the instructor.

**GROUP: 1**

**Members Present**:

|  |  |
| --- | --- |
| 1. Gyeongrok, Oh | 4. Pui Wai, Tse |
| 2. Hoi Kit, Cheung | 5. Sau Ching Yuki, Wong |
| 3. Lap Chi, Wong | 6. Yonghun Won |

## Milestone 2 Tasks

Some of the software for the project has already been written for you and is available on Blackboard. You must use this in your project and every team should add it to the source code for their repository. Anything in the main function is simply for demonstration purposes and can be replaced. The software you are being given has not been tested and you will need to test it.

You need to study the problem and the code provided for you and then:

* Add any new data structures you will require This will require a thorough analysis of the problem and the existing software. This should be done by creating a new header file in the directory where the rest of the source code has been placed. You do not want to go back and modify it later if you can avoid it as it will slow the project.
* Create a test plan for the project by replacing the text in the supplied test plan template with your test plan.

**Deliverables Due at End of Lab**

* Completed SCRUM report & reflections

**Deliverables Due within 48 hours of lab**

* An analysis of the problem (no written artifacts produced),
* A series of data structures created as header files and stored in the repository,
* A test plan stored in the repository.

**Rubric**

|  |  |  |
| --- | --- | --- |
| Individual | Group Participation | 75% |
| Teamwork | 10% |
| SCRUM Report | 15% |
| Group | Data structures (complete, correct and well-designed) | 20% |
| Test Plan (complete, well-written) | 20% |
| Git Usage (used properly with good structure) | 10% |
| Jira Usage (creates issues, tracks progress) | 10% |
| Meets Deadlines | 15% |
| SCRUM Report and Reflections | 25% |

**SCRUM Report**

**Summary of Tasks Completed or Delayed in the last week:**

Here you can list all of the tasks completed in the last week along with any tasks which could not be completed with a reason why they could not be completed.

|  |  |  |
| --- | --- | --- |
| **Member** | **Tasks Completed** | **Tasks Delayed/Blocked** |
| Gyeongrok, Oh | Reflections, scrum report | None |
| Hoi Kit, Cheung | Reflections, scrum report | None |
| Lap Chi, Wong | Git account creation, Jira account creation, Reflections | None |
| Pui Wai, Tse | Reflections, scrum report | None |
| Sau Ching Yuki, Wong | Reflections, scrum report | None |
| Yonghun Won | Reflections, scrum report | None |
|  |  |  |

For every task delayed or blocked, describe the reason for the delay or block, how it impacts the project and the proposed solution or workaround**.**

|  |  |
| --- | --- |
| **Delayed or Blocked Task** |  |
| **Reason for delay or block** |  |
| **Impact on Project** |  |
| **Solution or work-around** |  |
|  |  |
| **Delayed or Blocked Task** |  |
| **Reason for delay or block** |  |
| **Impact on Project** |  |
| **Solution or work-around** |  |

**Summary of Meeting:**

A summary of the main points discusses in the meeting and the outcomes of the discussions.

|  |  |  |
| --- | --- | --- |
| Topic | Discussion Summary | Outcome |
| Workload distribution | To distribute the tasks for milestone 2 to team members so that each member has a fair share of workload | The workload distribution is agreed in the meeting |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

**Summary of Decisions Made:**

This will include major architecture and design decisions, testing decisions, prioritization of tasks, dealing with problems encountered and other major outcomes from the meeting.

|  |  |
| --- | --- |
| Decision | Rationale |
| The requirements for creating data structures and writing test plan will be analyzed first before delegating the tasks | It is necessary to capture all requirements first to identify the tasks required for subsequent workload distribution. |
| The tasks are created in Jira to share the workload equally among team members. | Jira is a good project management platform for the team to keep track of the progress of each task. |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

**Tasks Attempted During Meeting:**

Each member is assumed to participate in the SCRUM meeting and contribute to the completion of the SCRUM report and reflections. Since the SCRUM meeting will not take more than 20-30 minutes, there is lots of time left to undertake some of the actual work tasks. In the table below, each member should list what they did to complete the SCRUM report, the reflections, and 1-4 other tasks they completed during the class period. If a task could not be completed, the student should indicate why this was not possible.

|  |  |  |  |
| --- | --- | --- | --- |
| Member | Task Attempted | Time Spent | Complete? |
| Gyeongrok, Oh | Reflections, code review | 30 mins | done |
| Hoi Kit, Cheung | Reflections, code review | 30 mins | done |
| Lap Chi, Wong | Reflections, code review | 30 mins | done |
| Pui Wai, Tse | Reflections, code review | 30 mins | done |
| Sau Ching Yuki, Wong | Reflections, code review | 30 mins | done |
| Yonghun Won | Reflections, code review | 30 mins | done |
|  |  |  |  |

**SCRUM Tasks Selected for Next Week**:

The tasks each member has selected to pursue for this class or the next week.

|  |  |
| --- | --- |
| Group Member | Task Description |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

**Major Outcomes of Meeting:**

This is where you should highlight the major accomplishments of the class.

|  |  |
| --- | --- |
| Outcome | Impact on Project |
| Requirements of data structures and test plan analyzed | Sub-tasks created to share the workload equally among team members and meet the requirements |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

**Things That Went Well in This Meeting:**

Here you can highlight things which worked well. This indicates that the way you worked on these items is working and should be continued.

|  |  |
| --- | --- |
| Topic/Work Item | Reason for Success |
| Workload distribution | The requirements of the data structures and test plan are fully understood by all team members to create reasonable sub-tasks |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

**Things That Did NOT go Well in This Meeting:**

This is where you can list things which did not go well in the class. You should analyze why this happened and suggest how you can improve it next time. This will lead to the goal of *continuous process improvement*.

|  |  |
| --- | --- |
| Topic/Work Item | Reason for Problem and How to do Better |
| NIL |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

**Reflections**:

1. In this milestone you have been asked to analyze a problem and design software(functions) to complete the solution without actually writing the software.
   1. Is this process more difficult than just writing the software to complete the project? If so, why is it more difficult? If not, why is it easier than just writing the software?  
        
      As every software writing requires analyzing the problem and designing software beforehand, we will assume we are comparing the difficulty of the following 2 situations when answering this question:  
      (1) Analyze the problem and designing software without writing the software  
      (2) Just write the software itself after someone has already done (1)  
        
      In our opinion, (1) is more difficult because it requires a deep understanding of the problem with the client, and then define the detailed business requirements. Based on these business requirements, corresponding system functions are required to be designed that should fully meet the business requirements. This involves a process of breaking down the problem into smaller components, defining their relationships and identify the most efficient way to solve the problems. We must also think of the possible extreme cases that may not be visible at first sight, and ensure the software design can cater these possible scenarios. Such activities require a higher-level of abstract thinking as compared to just writing the software itself, where the requirements are already provided in a detailed manner. Problem analysis and software design also need a higher level of collaboration and communication skills among all stakeholders, i.e. client, developers, testers and quality assurance team, whereas developers probably only need to focus on the program code.
   2. Describe two advantages of developing software in this manner rather than just moving on to writing the functions without writing specifications first.

Firstly, a comprehensive understanding of the project requirements can be obtained. Since the requirements of a big solution involving teams to accomplish should be complicated and the potential risks are non-trivial, discovering the problems and solutions before directly writing the software can highly reduce the cost of modifications due to misunderstanding of requirements afterwards.

Secondly, the quality of the final deliverable can be controlled. Having the well-defined specifications, the future development process can have a standard to refer to without shifting away from the objectives. For testing, test plan can be created according to the specifications so that there are definite test cases and expected results. For release, the level of completion can be quantified to assess whether the solution is ready to be released.

1. Why is it a good idea to create a test plan? Describe at least 3 advantages of test plans.  
     
   First, a test plan provides a structured approach to testing by outlining the objectives, scope, and strategies for testing. It helps testers and stakeholders have a clear understanding of what needs to be tested, how it will be tested, and the expected outcomes. By having a well-defined test plan, testing activities can be organized, prioritized, and executed efficiently, ensuring that all necessary aspects of the software are thoroughly tested.

Second, a test plan helps ensure comprehensive test coverage by identifying the test scenarios, test cases, and test data needed to validate different aspects of the software. It allows testers to map requirements and functional specifications to corresponding tests, enabling traceability between the tests and the software's expected behavior. This helps in identifying any gaps in coverage and ensures that all critical functionalities and use cases are tested, reducing the risk of potential issues in the production environment.

Third, a test plan facilitates risk management by identifying potential risks. It allows testers to assess the impact and various of risks on the testing process. By proactively addressing risks in the test plan, such as resource constraints, time limitations, or dependencies on external systems, appropriate measures can be taken to minimize their impact. This proactive risk management approach helps in reducing the probability of testing delays, failures, or unexpected issues, thus ensuring a smoother testing process.

Fourth, a test plan promotes effective communication and collaboration among the testing team, stakeholders, and other project members. It serves as a reference document that can be shared and discussed with the relevant parties, ensuring everyone is aligned on the testing approach and objectives. By having a shared understanding of the test plan, stakeholders can provide feedback, raise concerns, and make informed decisions regarding the testing process. This collaborative approach fosters transparency, reduces misunderstandings, and enhances the overall quality of the testing effort.

1. Describe the process you used to analyze and understand the existing software.

Gather all available documentation and collect feedback from users and stakeholders to understand the software's functionalities and requirements.

Conduct interviews with developers, testers, and end users to gain deeper insights into the software's purpose, functionality, issues, and usage contexts.

Perform a code review to understand the software's structure, organization, and implementation details, paying attention to coding patterns, frameworks, and libraries used.

If access to the source code is limited, employ reverse engineering techniques using specialized tools to gain insights into the software's functionality and behavior.

Conduct system analysis to understand the overall architecture, identify components and modules, and determine data flows and integration points with external systems.

Analyze the software's functional requirements, comparing them to the implemented functionality to identify any gaps or discrepancies and understand its features, use cases, and user interactions.

Evaluate non-functional aspects such as performance, scalability, security, and usability to identify any bottlenecks, vulnerabilities, or usability issues that need to be addressed.

Review existing documentation to identify gaps or inconsistencies and update it as necessary to ensure accuracy and completeness.

Develop a testing strategy, create test cases and scenarios, and execute tests to validate the software against its requirements and identify defects or issues.