# **MILESTONE 3** -- SFT221 SCRUM Report and Reflection

All students are expected to attend the SCRUM meetings and to participate. Failure to do so will result in greatly reduced grades.

**GROUP**: \_\_\_\_\_\_\_\_\_\_2\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Members Present**:

|  |  |
| --- | --- |
| 1. Johnny | 4. Theo |
| 2. Noah | 5. Ronnato |
| 3. Armaan | 6. |

## Milestone 3 Tasks

In this milestone you will create issues to design the functions, design all of the functions you need to complete the project and store the specifications in the repository. As soon as the specifications start to be produced, you can start to design the blackbox tests (what they test, how to perform them and test data). Once tests are written, they can be implemented and added to the repository and any team members not otherwise busy can start to implement the functions. You will also build a function-test matrix that shows the blackbox tests for each function. This will be maintained through the testing cycle as new tests are added.

**Deliverables due 4 days after your lab day:**

* A set of AT LEAST 4 function specifications added to a new header file and stored in the repository.
* A set of blackbox tests as test documents (in an Excel file) with test data for the functions you created. At least 4 sets of test data are required for each function. You must have test cases for at least 6 functions (including all your custom function). Stored in the repository.
* **Create and add a C++ testing project to your solution.**
* Start writing blackbox test code (for the functions above) and store in repository (at least 1 is required for this milestone).
* Start implementing the functions and store them in repository (optional).
* A requirements traceability matrix added to the repository and shows the mapping between the requirements and test cases.
* Updated Jira project to show activities and progress.
* Completed scrum report including reflection questions answered.

**Rubric**

|  |  |  |
| --- | --- | --- |
| **Individual** | Group participation (includes GitHub commits and Jira usage) | 80% |
| Teamwork | 20% |
| **Group** | Function specifications (documented, complete, well-written, added to the project) | 10% |
| Blackbox test cases document (well-written, complete, good test data) | 15% |
| Blackbox test code (in the C++ project) well-designed and documented | 15% |
| Functions implementation (coded in the C project & well documented) | 10% |
| Requirements traceability matrix (complete and added to GitHub) | 10% |
| Git usage (used properly with good structure) | 10% |
| Jira usage (creates issues, tracks progress) | 10% |
| Scrum report & reflections | 20% |
| **Deadline** | 20% deduction for each day you are late |  |

**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** |
| **Theo** | **Completed Presentation** | **n/a** |
| **Noah** | **SCRUM, Contract** | **n/a** |
| **Armaan** | **SCRUM, Contract** | **n/a** |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

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

**Summary of Meeting:**

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

|  |  |  |
| --- | --- | --- |
| Topic | Discussion Summary | Outcome |
| Scrum | **Scrum report** | **Prepared scrum report** |
| Test plan | **Planning test** | **Creating plan outline** |
| Presentation | **Preparing presentation** | **Prepared presentation** |
| Jira | **Getting program** | **completed** |
|  |  |  |
|  |  |  |
|  |  |  |

**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 |
| Tasks division | Equal division of work load to each member. |
| Testing paths | Shortest Path Calculation, Capacity Calculation, Output Message Generation, Shipment Allocation |
| Presentation created | Created the presentation in accordance to all the milestones |
|  |  |
|  |  |
|  |  |
|  |  |

**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? |
| ALL | **Discussed problem and possible solutions/resolutions to the situation** | **45 min** | **Yes** |
| ALL | **Answering Scrum report** | **1 hr** | **Yes** |
| ALL | **Discussion and creating Test Plan** | **1 hr** | **Yes** |
| ALL | **Creating the group presentation** | **1 hr** | **Yes** |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**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 |
| ALL | Scrum/Reflection questions |
| ALL | Creating plan for milestone 5 |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

**Major Outcomes of Meeting:**

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

|  |  |
| --- | --- |
| Outcome | Impact on Project |
| Created group presentation | **Created group presentation and identified bugs and did black box testing** |
| Test planning created | **Set testing specifications as a group** |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

**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 |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

**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 |
| Scrum | **Questions answered through discussions** |
| Presentation | **Discussed the topics for presentation and completed it on time** |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

**Reflections**:

Answer the following questions using your own words. Make sure that each answer comprises a minimum of 100 words.

1. In this milestone, we write the blackbox tests but not the whitebox tests. Explain why we can write the blackbox tests but not the whitebox tests.

Blackbox tests are appropriate for early-stage testing when code details may not be reliable since they concentrate on the software's functioning without examining its internal code. Whitebox tests are usually carried out later in development, when the codebase is more polished, and they necessitate familiarity with the inside code. Consequently, compared to whitebox tests, blackbox tests can be created sooner in the process.

Blackbox testing and whitebox testing are two different approaches to software testing, each with its own methodology and objectives.

Blackbox Testing:

In blackbox testing, testers examine the functionality of a software application without knowing its internal workings. They focus solely on the inputs and outputs of the system, treating it as a "black box" where they cannot see the internal structure or code.

Testers design test cases based on specifications, requirements, and user expectations. They simulate various inputs to see how the system responds and verify whether the outputs are correct.

Blackbox testing is primarily concerned with validating the software against its intended behavior and requirements, rather than its implementation details.

Testers can write blackbox tests early in the development process because they don't require knowledge of the internal code. They can start testing based on the functional specifications or user stories provided.

Whitebox Testing:

In whitebox testing, testers have access to the internal structure, code, and logic of the software. They design test cases based on this knowledge, aiming to exercise different paths, conditions, and branches within the code.

Whitebox testing is concerned with verifying the correctness of the code implementation, including edge cases and error handling, as well as ensuring code coverage.

Writing whitebox tests typically requires a good understanding of the codebase and its intricacies. Testers need to analyze the code to identify potential weaknesses and areas that need testing.

Unlike blackbox testing, whitebox testing often occurs after the code has been written or during the later stages of development when the codebase is stable and accessible for inspection.

Now, to address your question:

In the milestone mentioned, the focus is on writing blackbox tests but not whitebox tests. There could be several reasons for this:

Early Stage of Development: At this stage, the software may not have reached a point where its internal codebase is stable or complete enough for whitebox testing. Developers might still be actively coding and iterating on features.

Requirement and Specification Focus: Blackbox testing is well-suited for validating the software against its requirements and specifications. Therefore, it's prioritized to ensure that the software meets its intended functionality and user expectations.

Testing Independence: Blackbox tests can be written independently of the code implementation. Testers can begin validating the software's behavior without waiting for the entire codebase to be completed or fully understood.

Resource Allocation: Resources such as time, personnel, and tools may be allocated primarily for blackbox testing at this stage, with the intention of conducting whitebox testing later in the development cycle when the codebase is more stable.

In summary, while both blackbox and whitebox testing are essential components of a comprehensive testing strategy, the decision to prioritize blackbox testing over whitebox testing in a specific milestone could be due to factors such as the stage of development, testing objectives, and resource allocation.

1. Explain why we need the function-test matrix and why it is important in a large project.

A function-test matrix is crucial in large projects for several reasons:

Coverage analysis: It assists in making sure that every feature and function of the software has been sufficiently tested. Teams can find any gaps in test coverage by using the matrix, which maps functions to matching test cases. This guarantees thorough testing and lowers the possibility of undetected flaws making their way into the finished product.  
  
Test Management and Planning: The matrix acts as a roadmap for test management and planning. By identifying the functions that require testing and the test cases that address those functions, it aids teams in planning their testing activities. This guarantees that testing activities are efficient, organised, and concentrated on important programme features.

Traceability: It provides traceability between requirements, functions, and test cases. This means that each test case can be linked back to specific requirements or functions, allowing for better visibility into the testing process. Traceability helps stakeholders understand how well the software meets its requirements and ensures that all requirements are adequately tested.

Risk Assessment: The function-test matrix facilitates risk assessment by highlighting high-risk functions or features that require more rigorous testing. Teams can prioritize testing efforts based on the criticality of functions, ensuring that resources are allocated effectively to mitigate potential risks.

1. Other life cycle models left team members idle while waiting for parts of the project to be completed. Describe how an agile model, like the one we are using, avoids this problem and keeps the whole team busy all the time. Does this make managing the project simpler or more complex and why?

The iterative and incremental approach used in agile methodologies, like Scrum, Kanban, or Extreme Programming (XP), guarantees team members' ongoing engagement and productivity throughout the project lifespan. Here's how agile approaches keep team members engaged and prevent them from being idle:

1. Iterative Development: Agile projects are divided into short iterations or sprints, typically ranging from one to four weeks. Each iteration delivers a potentially shippable product increment. This allows team members to focus on completing specific, manageable chunks of work within a short timeframe, minimizing idle time.
2. Cross-Functional Teams: Agile teams are typically cross-functional, consisting of members with diverse skills and expertise. Team members collaborate closely on all aspects of the project, including planning, development, testing, and deployment. This ensures that there's always work to be done across different areas, keeping everyone busy and engaged.

Agile approaches encourage teamwork, flexibility, and productivity, but they can also make project management more difficult:

Dynamic Nature: Agile projects are highly dynamic and require constant monitoring and adaptation. Project managers need to continuously prioritize and reprioritize tasks, adjust timelines, and manage stakeholder expectations in response to changing requirements and feedback. This dynamic nature can make project management more complex compared to traditional, plan-driven approaches.

Communication and Coordination: Agile projects rely heavily on communication and collaboration among team members. Project managers need to facilitate effective communication, ensure alignment among team members, and resolve any conflicts or bottlenecks that may arise. Managing the coordination and synchronization of activities across a cross-functional team can be challenging, especially in large and distributed teams.