# **Milestone 3 Scrum Report**

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

**GROUP**: \_5\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Members Present**:

|  |  |
| --- | --- |
| 1. Phuong Bac Nguyen | 4. Cristian Vargas |
| 2. Shuja lashkari | 5. |
| 3. Maryam Jawed | 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) | 10% |
| Blackbox test code (in the C++ project) well-designed and documented | 10% |
| Functions implementation (coded in the C project & well documented) | 10% |
| Visual Studio solution with 2 projects (complies and works) | 10% |
| Requirements traceability matrix (complete and added to GitHub) | 10% |
| Git usage (used properly with good structure) | 10% |
| Jira usage (creates issues, tracks progress) | 15% |
| Scrum report & reflections | 15% |
| **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** |
| Phuong Bac Nguyen | requirements traceability matrix |  |
| Phuong Bac Nguyen | 4 function specifications |  |
| Maryam Jawed | Blackbox test cases document (well-written, complete, good test data) |  |
| Phuong Bac Nguyen | function description template |  |
| Phuong Bac Nguyen | Create and add a C++ testing project to the solution. |  |
| Phuong Bac Nguyen | Visual Studio solution with 2 projects (complies and works) |  |
| Cristian Vargas - Shuja lashkari | Scrum report & reflections |  |

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 |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

**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 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? |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**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 |
| Pending | Complete the implementation of all remaining functions in `functions.c |
| Pending | Design and execute white-box tests, documenting results in `white-box-tests.xlsx` |
| Pending | Run all unit and integration tests, identify failures, and debug code |
| Pending | Set up a pre-push Git hook to automate test execution before pushing changes |
| Pending | Update the Scrum Report (`SCRUM\_Milestone4.md`) with completed tasks and issues found |
| Pending | Ensure all new test cases and fixes are correctly pushed to GitHub and documented in Jira |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

**Major Outcomes of Meeting:**

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

|  |  |
| --- | --- |
| Outcome | Impact on Project |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

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

**Reflections**:

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

1. How did your approach differ when creating test cases for blackbox testing versus whitebox testing? Reflect on the advantages and limitations of each method based on your experience with the assignment.

Shuja lashkari: When creating test cases for black-box testing, I focused on the system’s external behavior such as the user perspectives to design inputs and expected outputs without knowing the internal code structures. This approach felt intuitive and user-focused, but it sometimes missed hidden logic errors or edge cases. For white-box testing, I dove into the code, ensuring every path and condition was tested, which felt thorough but required more technical effort and time. Both methods have their strengths—black-box keeps the user in mind, while white-box ensures the code is solid—but each has its blind spots, making them both crucial.   
Cristian: While black-box testing ensures the system meets user expectations, white-box testing provides deeper insight into the internal logic. One key advantage of white-box testing is identifying redundant or inefficient code, which helps optimize performance. Additionally, it ensures that all conditions and branches are executed, reducing the risk of hidden bugs. Combining both methods improves overall software quality by validating functionality and implementation efficiency

1. How did the traceability matrix help ensure that all functional specifications were adequately tested? Reflect on its role in maintaining comprehensive test coverage.

Shuja lashkari: The traceability matrix ensured all functional specifications were tested by linking requirements to test cases, making it easy to track coverage and identify gaps that could’ve been overlooked. It provided clarity especially when requirements changed, and helped verify that every functionality was validated by acting as a safety net.

Cristian: In addition to linking requirements to test cases, the traceability matrix also helps prioritize testing by identifying critical functionality. It ensures that changes to requirements do not leave untested areas, maintaining consistency throughout the development cycle. It also serves as documentation for debugging, making it easier to trace failed test cases back to specific requirements. This structured approach improves test coverage and minimizes missed issues.

1. Write down two of the function prototypes you submitted. Why did do you need each one of them and how will each one help you achieve the project needs?  
   Shuja lashkari:

checkWeight this function ensures the shipment weight is within acceptable limits (1 to TRUCK\_MAX\_WEIGHT), preventing invalid or unsafe shipments. It helps by validating the weight, it ensures compliance with project requirements and avoids errors in processes like shipping calculations or truck loading. int checkBoxSize Why it’s needed, this function verifies if the box dimensions are valid, ensuring the box meets size constraints for storage or transportation. How it helps: It ensures only properly sized boxes are processed, maintaining efficiency and preventing issues such as storage systems.  
Another critical function is isValidDestinationFormat, which ensures that delivery addresses follow the correct format before processing. This prevents errors in package routing and avoids unnecessary reassignments. Additionally, checkDestination verifies that the specified address exists within a valid building, ensuring deliveries are only assigned to reachable locations.