# 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**: \_\_\_\_E\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Members Present**:

|  |  |
| --- | --- |
| 1. Jay Vijaykumar Vakil | 4. Hetav Mamtora |
| 2. Harsh Dugar | 5. Viren Vaishnav |
| 3. Mohit Sheth | 6. Srujal Patel |

## Milestone 5 Tasks

In this milestone, you should write, implement, and execute integration tests. Integration tests test how multiple functions work together to complete a task. Depending on what is being tested, you might be able to write unit tests to do the testing and automatically compare the results. In other cases, you might need to manually check the output to check it. This will all be stated in the tests where it discusses how they should be run.

As you update the function-test matrix, you will need to add a very brief description for each integration test so the matrix will clearly show what the tests are testing. Acceptance tests will be tested against actual user requirements and will list all the tests for each requirement.

Acceptance tests are the final tests and are largely aimed at showing the customer that the correct output is produced for different inputs. This will largely require manual testing.

**Deliverables Due at end of Lab:**

* Completed SCRUM report and reflections

**Deliverables Due at 23:59 12 Days after Lab:**

* integration tests written and stored in repository,
* integration tests written (store in repo), executed (results in Jira and in test documents) and debugged.
* acceptance tests written and stored in repository.
* Updated function-integration-requirements-test matrix stored to the repository.

**Rubric**

|  |  |  |
| --- | --- | --- |
| Individual | Group Participation | 75% |
| Teamwork | 10% |
| SCRUM Report and reflections | 15% |
| Group | integration tests (well-designed, written and documented) | 20% |
| acceptance tests (well-designed, written and documented) | 20% |
| Test Execution (performed, results recorded, issues created) | 15% |
| Debugging (Bugs fixed, documented, Jira updated) | 5% |
| Function-test matrix updated | 5% |
| Git Usage (used properly with good structure) | 5% |
| Jira Usage (creates issues, tracks progress) | 5% |
| Meets Deadlines | 5% |
| SCRUM Report and Reflections | 20% |

**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** |
| **Jay Vijaykumar Vakil** | **Implemented and executed the project code, Made Unit tester project—linked Unit testers. Helped implement test cases.** | **None** |
| **Mohit Sheth** | **Made the implementation of the unit test cases both white and black. Helped completing the SCRUM report** | **NONE** |
| **Harsh Dugar** | **Made white box test cases for one function and Also made and updateed the Traceability matrix for both the tests, whitebox and blackbox.** | **None** |
| **Hetav Mamtora** | **Made Whitebox test cases as well as helped in creating tester and helped throughout with any issues.** | **None** |
| **Srujal Patel** | **Answered one of the reflection questions in the scrum report.** | **None** |
| **Viren Vaishnav** | **Scrum report reflection question.** | **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** | **None** |
| **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 |
| Further requirement of milestone | **Everyone came up with points as well everyone was given a day to come up with what they want to do** |  |
| Discuss integration and acceptance tests | **After the review of requirements three of the guys were part of creating an integration and acceptance tests.** |  |
| Implementation of test code and execution | **Two of the guys decided to take up running and implementation of code** |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

**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 |
| To come up tests |  |
| To run them and work in implementation |  |
| To update the traceability matrix |  |
| To finish the scrum and submit |  |
|  |  |
|  |  |
|  |  |

**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? |
| Jay Vijaykumar Vakil | **Made function implementation that assigns truck.** |  | **YES** |
| Mohit Sheth |  |  |  |
| Hetav Mamtora | **Helped in tester creation and gave overall requirements of number of test cases and description of tests.** |  | **Yes** |
| Harsh Dugar | **Helped in completing the reflection answers and assigning tasks with the leader.** |  | **Yes** |
| Srujal Patel | **Analysed the function given by my mates and tried to create the whitebox test cases for those functions.** |  | **YES** |
| Viren Vaishav | **Finding the requirenment of the test cases and get the review of overall project** |  | **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 |
| Jay Vijaykumar Vakil | Complete the remaining implementation and unit testers. Help team to come up with new test cases |
| Hetav Manojkumar Mamtora | Ongoing with more test cases as well helping in tester creation and implementation. |
| Viren Vaishnav | Made test cases for black box testing and white box testing |
| Harsh Dugar | Updating the Traceability matrix and making acceptance tests. |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

**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 |
| Everyone decided what they were gonna do and did it | **Completion of tasks were chosen.** |
| Implementation and creation | **Equally divided and well collaboration among the group.** |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

**Things That Did NOT go Well in This Meetingll**

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**:

1. At this point, you are using the GIT hook to automate testing. Have you found that any of the tests failed and prevented you from pushing your code to the repository? If so, how did you handle the situation?

While using Git hooks for automated testing we encountered test failure that prevents pushing code to repository initiates a systematic resolution process. to resolve this, we promptly dive into the test results to gain insight into the root cause of the failures. We start by examining error messages and logs generated during the automated tests. This careful analysis helps us pinpoint the issues that need to be addressed. we look for logic errors, edge cases that were not adequately handled, and potential integration problems that might have led to the test failures.

With a clear understanding of the problems, we proceed to fix the code. we make the necessary adjustments, ensuring that we address all the issues identified during the test failure analysis. Once the code changes are made, we rerun the local tests to verify that the fixes have been successful.

If, during the pre-push hook execution, any tests still fail, the push is automatically blocked. At this point, we return to the code and repeat the resolution process. we continue iterating through the cycle of identifying the issues, fixing the code, and rerunning the tests until all the tests pass successfully during the pre-push hook execution.

1. Explain why we are automating the testing process and what the advantages of this automation are.  
   Automating the testing process is crucial due to its numerous advantages. Firstly, automation reduces manual effort and saves time, enabling rapid and frequent testing, which is essential in agile development. Secondly, it enhances accuracy by eliminating human errors and ensuring consistent test execution. Thirdly, automated tests can be easily reused, reducing redundancy and improving overall test coverage. Moreover, automation allows early detection of defects, leading to faster bug fixing and better software quality. It facilitates continuous integration and delivery, streamlining the development cycle. Ultimately, automation boosts productivity, cost-effectiveness, and overall efficiency of the testing process.

1. Did you find the integration and acceptance tests more difficult to write than the black box and white box tests? If so, why were they harder to write? Did you write more white box and black box tests or more integration and acceptance tests?  
     
     
     
     
   Coordination tests have a few advantages over black box and white box tests because they focus on a different part of the product improvement process. A few advantages of taking tests include the following:

Joining tests check that association focuses and relationship between parts capability appropriately. This is particularly significant in light of the fact that defects regularly happen at the limits of where units speak with each other.

Gatherings can screen the item's security as new elements and adjustments are added by routinely driving consolidation tests. This perceives lose the faith gives without a second thought in the improvement cycle.

Performance and scalability tests can be part of integration tests to make sure the integrated system can handle expected loads and keep acceptable response times.

Data integrity, error handling, and security are among the difficult-to-cover cross-cutting issues covered by integration tests.

Mix tests are more qualified to endorse complex business streams that range different units, helping with guaranteeing that the whole work process is working as per its particulars.

While mix tests offer these advantages, it's basic to see that they supplement black box and white box tests rather than supplanting them. To guarantee the item's overall quality and dependability, a total testing strategy combines every one of the three levels of testing — unit, consolidation, and affirmation.

1. Explain why it is necessary to write integration and acceptance tests given that all of the code has already passed black box and white box tests.

Even if the code has already passed the black box and white box tests, integration and acceptance tests are still required to ensure that the software meets the desired quality and functionality. We should look at why these tests are fundamental:

Black box testing centers around approving the usefulness of the product from an end-client point of view disregarding its inner execution. It guarantees that the product meets the predetermined prerequisites.

White box testing, then again, analyzes the inner design and rationale of the product to confirm that all code ways are executed and to find expected mistakes at the code level.

Black box and white box tests typically focus on individual units (such as functions, classes, and modules) at the unit level. They confirm whether every unit proceeds as expected in segregation.

As the name suggests, integration tests evaluate how various components interact with one another and the integrated system as a whole. This is significant in light of the fact that imperfections can emerge when units collaborate and trade information.

The goal of acceptance tests is to make sure that the system meets the requirements and meets the expectations of the users. They assist with guaranteeing that the framework fulfills the general business needs.

New issues that were not apparent during unit testing may surface during integration. Interactions between components that were not apparent in isolation may be to blame for these issues.

Issues that are only apparent from the user's perspective and may not have been identified during the development process can be discovered through acceptance tests.

Combination and acknowledgment tests are fundamental to survey the general client experience and whether the product meets the expected business objectives.

While unit tests check individual functionalities, incorporation and acknowledgment tests look at the product as a total framework, guaranteeing that it capabilities flawlessly and meets the end-clients' assumptions.

Leading every one of the three degrees of testing (unit, joining, and acknowledgment) gives a more extensive approval of the product's rightness, strength, and usefulness.

The mix of these tests imparts trust in the quality and soundness of the product before it is sent to creation.

Even if the software passed previous tests, new features and code changes could cause problems. Incorporation and acknowledgment tests assist with getting relapses and guarantee that the recently added functionalities don't adversely affect existing elements.

All in all, joining and acknowledgment tests supplement black box and white box tests by surveying the framework all in all and approving that it meets both practical necessities and client assumptions. Each sort of testing assumes a one of a kind part in guaranteeing the product's quality and dependability. Skipping combination and acknowledgment tests could prompt undetected issues, possibly imperiling the general progress of the undertaking.