# **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**: \_\_\_\_\_\_\_\_\_\_4\_\_\_\_\_\_\_\_\_\_\_\_

**Members Present**:

|  |  |
| --- | --- |
| 1. | 4. |
| 2. | 5. |
| 3. | 6. |

## 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** |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

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

**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**:

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?  
        
      Both processes of analyzing a problem and designing the solution versus writing the software have their own unique challenges. Problem analysis and solution design requires you to break down the problem into more manageable parts and to carefully consider what those parts need, while keeping in mind how the parts all integrate in the end. Making these decisions can be difficult. On the other hand, writing software requires a good understanding of the program language’s rules and syntax, attention to detail, and problem-solving skills. Neither seem more difficult than the other, especially from a team perspective. For example, some team members may find understanding the problem difficult, while other members may find remembering code syntax more challenging.

However, it is important to note our team may initially find writing software collaboratively more difficult since we have never been required to do so for our program until now. Additionally, some team members are not familiar with GIT and GitHub, and until they overcome the learning curve, we may encounter difficulties. Once we get familiar with these tools and establish a comfortable workflow, writing software as a team will be much easier.

* 1. Describe two advantages of developing software in this manner rather than just moving on to writing the functions without writing specifications first.  
       
     The first advantage of developing software in this manner is a cohesive understanding among team members. Analyzing the problem and identifying data structures before moving on to writing functions ensures everyone has a clear understanding of the problem, the solution requirements, and the goals of the project. This minimizes misunderstandings, debugging time, and rework down the line, which leads to better collaboration and more efficient workflow. It makes developing a solution as a team easier and faster and encourages clean, DRY (“Don’t Repeat Yourself”) code.

The second advantage is efficient resource management. By identifying the specifications of the problem first, our team can divide and allocate tasks, time, and other resources more efficiently to satisfy the specifications. This approach allows our team to prevent unnecessary time and effort costs throughout the software development process.

1. Why is it a good idea to create a test plan? Describe at least 3 advantages of test plans.

Effective software testing requires the creation of a test plan. The following three benefits of having a thorough test plan:

1. Wide-ranging Protection: Test plans are helpful, to ensure thorough test coverage. They explain the various kinds of testing that must be done, including functional, performance, security, and usability testing, among others. A test plan guarantees that various parts of the software are adequately tested by determining the right test techniques and procedures. This thorough coverage aids in the discovery of flaws, weaknesses, and usability problems, resulting in a software solution that is more robust and dependable.
2. Risk Assessment and Mitigation: The identification and reduction of risks related to the software development and testing process are made easier by a test strategy. It aids in identifying dependencies, bottlenecks, and major areas that need more attention. This is very similar to section 5.c which highlights the severity levels of issues. It is possible to successfully limit these risks by analyzing them in advance. For instance, extra resources or specialized testing methods can be assigned if a test strategy identifies a high-risk region to ensure complete testing in that domain. The possibility of significant problems occurring during or following the software release is decreased by using this proactive risk management strategy.
3. Logical approach: An organized and structured approach to testing is provided by a test plan. To ensure that testing operations are carried out methodically, it describes the objectives, scope, and test strategies. Following a predetermined plan makes testing more effective and decreases the likelihood of missing crucial test cases or criteria.

 Overall, a test plan serves as a guide for testing efforts, giving them a defined direction, guaranteeing thorough coverage, and reducing any risks. It improves the effectiveness of the testing procedure, encourages improved teamwork, and helps provide high-quality software.

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