# **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**: GROUP 2

**Members Present**:

|  |  |
| --- | --- |
| 1Faaz Sherwani 113026223 | 4.Iraklis Tsanachtsidis 122226228 |
| 2. Frank Prerez 141647222 | 5.Aum Rasikbhai Parsana 112872221 |
| 3. Tarun Thomas 113605224 | 6.Rutarj Mrushad Shah 170870216 |
| 7. . Jubril Olawale Akolade 167529213 |  |

## 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** |
| Whole Team | Analysis of Existing Software |  |
| Faaz | Created a New Header Files |  |
| Faaz | Add New Data Structures |  |
|  | Test Plan Creation |  |
| Jubril & Iraklis | 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**.**

|  |  |
| --- | --- |
|  | **-There was no delayed task for this milestone** |
| **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 |
| Design | The meeting focused on the tasks related to Milestone 2, including adding new data structures to the existing software and creating a test plan for the project. | New data structures were identified and defined in a new header file in the source code directory. |
|  |  |  |
| Analysis and planning | The team analyzed the provided code and identified areas where additional data structures would be beneficial for enhancing the software's capabilities and improving performance. | The team successfully implemented the new data structures and ensured their integration with the existing codebase |
|  |  |  |
| Header creation | A new header file was created in the source code directory to define the new data structures. | Completion of Header file |
|  |  |  |
| Implementation | The team implemented the new data structures, ensuring seamless integration with the existing codebase and following coding conventions. | Good team coordination |
| SCRUM implementation | The team documented their progress in the SCRUM report, including challenges faced, lessons learned, and plans for the next iteration. | Documentation of the events |

**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 |
| Adding new data structures to the existing software. | The analysis of the existing software indicated the need for additional data structures to enhance the software's capabilities and improve performance. |
|  |  |
| Creating a new header file to define the new data structures. | By creating a separate header file, the team ensures modularity and ease of maintenance, as well as adhering to good software engineering practices. |
|  |  |
| Documenting progress in the SCRUM report. | The SCRUM report serves as a record of the team's progress, challenges, and lessons learned, aiding in project management and future planning. |
|  |  |
|  |  |

**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? |
| Jubril & Iraklis | Completed the SCRUM report by documenting the team's progress during Milestone 2, including tasks completed, challenges faced, and plans for the next iteration. | 1 hour | Yes |
| All members | Participated in the discussion to analyze the existing software and identify areas where new data structures would be beneficial. | 45 min | Yes |
| All Members | Contributed to the design and implementation of the new data structures, ensuring their integration with the existing codebase. | 30 min | Yes |
| All member | Test Plan for next weeks milestone | 25 min | 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 Members | Conducting further testing and debugging of the new data structures to ensure their reliability and accuracy. |
|  |  |
| All Members | Assisting in the integration of the new data structures into the existing software, ensuring seamless functionality. |
|  |  |
| All Members | Collaborating with other team members to optimize the performance of the software using the newly added data structures. |
|  |  |
| All Members | Reviewing and refining the test plan to ensure comprehensive coverage of test scenarios and edge cases. |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

**Major Outcomes of Meeting:**

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

|  |  |
| --- | --- |
| Outcome | Impact on Project |
| Test Planning | **Coordination and planning** |
| Struct declarations | **Struct Creation** |
| Header File | **Creation of header files to continue the ccode** |
| SCRUM Report | **Completion of SCRUM report** |
|  |  |
|  |  |
|  |  |

**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 |
| Individual Work assignment | **Each member got their individual part** |
| Test planning for next milestone | **A plan is created so we know ahead how to tackle the problem** |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

**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 |
|  | Everything was great, and all went according to plan |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

**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?

Answer:

Analyzing a problem and designing software functions without writing the actual code can be more challenging than simply writing the software to complete the project. There are several reasons why this process can be more difficult:

1. Abstract Thinking: Analyzing and designing software requires abstract thinking and problem-solving skills, which can be more challenging than following a set structure and syntax while writing code.

2. Complexity Management: Analyzing a problem and designing software involves managing complexity, considering various factors like performance, scalability, maintainability, and usability. Handling this complexity can be more difficult than just writing code for a specific task.

3. Collaboration and Communication: Analyzing and designing software often involves collaborating with others and communicating ideas effectively, which adds another layer of complexity to the process.

4. Potential for Mistakes: Analyzing and designing software carries the risk of making mistakes that can have long-term consequences. Careful analysis and planning are necessary to minimize mistakes and ensure a solid foundation for the project.

Despite its challenges, this process is important as it helps identify issues early on, resulting in a more robust and efficient development process.

* 1. Describe two advantages of developing software in this manner rather than just moving on to writing the functions without writing specifications first.

Developing software by analyzing the problem and designing specifications before writing the functions offers several advantages over directly moving on to writing the code. Here are two key advantages:

1. Improved Clarity and Understanding: By taking the time to analyze the problem and design software specifications, developers gain a deeper understanding of the problem domain and the requirements. This process helps clarify the objectives, functionality, and desired outcomes of the software. As a result, the development team can align their efforts and work towards a shared understanding of the project, reducing the risk of misinterpretation or misunderstandings during the coding phase.

2. Enhanced Efficiency and Quality: Developing software with well-defined specifications allows for more efficient development and higher-quality outcomes. The specifications act as a blueprint or roadmap for the development process, guiding the implementation phase. By having a clear vision of what needs to be achieved, developers can streamline their coding efforts, avoid unnecessary rework, and reduce the risk of introducing errors or bugs. This approach promotes better code organization, modular design, and adherence to coding standards, ultimately leading to a more maintainable and scalable software solution.

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

Creating a test plan is crucial for ensuring the quality and reliability of a software project. Here are three key advantages of having a test plan:

1. Comprehensive Test Coverage: A test plan helps ensure comprehensive test coverage by outlining various scenarios, edge cases, and functional requirements that need to be tested. It provides a systematic approach to validating the software's functionality and behavior in different situations. With a test plan, developers can identify potential issues and bugs early on, reducing the risk of critical failures in the production environment.

2. Efficiency in Testing Efforts: Having a well-defined test plan helps optimize testing efforts. It allows testers to prioritize test cases based on their criticality and potential impact on the software's performance. By focusing on high-priority areas and critical functionalities, testers can efficiently allocate resources and conduct targeted testing. This approach saves time and effort by avoiding unnecessary or redundant tests, ensuring that testing efforts are focused on areas that matter most.

3. Consistency and Reproducibility: A test plan provides clear guidelines and instructions for conducting tests, ensuring consistency and reproducibility. Testers can follow predefined steps and test data, enabling consistent testing across different team members or testing cycles. Consistency in testing allows for easier identification of issues and more reliable results. Moreover, if a bug is discovered during testing, a well-documented test plan allows developers to reproduce the issue, understand its cause, and resolve it effectively.

Overall, a test plan provides the framework for organized and structured testing, leading to comprehensive test coverage, efficient testing efforts, and consistent and reproducible results. By having a well-designed test plan, software projects can achieve higher quality, improved reliability, and faster resolution of issues, contributing to overall customer satisfaction and success of the project.

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

To analyze and understand the existing software, the following process can be employed:

1. Initial Review: Start by conducting an initial review of the existing software codebase. Get familiar with the overall structure, file organization, and major components. This step provides a high-level understanding of the software's architecture and layout.

2. Identify Key Functionality: Identify the key functionality and features of the software. This involves identifying the main modules, classes, or functions that drive the software's core operations. Pay attention to any documentation or comments within the code that provide insights into the software's purpose and functionality.

3. Trace Program Flow: Trace the program flow by following the execution path through the code. Identify the entry points, major control structures (such as loops or conditionals), and the interactions between different components. This step helps in understanding the sequence of operations and the dependencies between different parts of the software.

4. Analyze Data Structures: Analyze the data structures used in the software. Identify the main data types, objects, or data models employed and how they are organized and utilized. Understand the relationships between different data structures and how they are accessed or manipulated within the code.

5. Review Algorithms and Logic: Examine the algorithms and logic implemented in the software. Understand the computational processes involved, such as sorting, searching, or data transformations. Identify any complex or critical algorithms that require special attention.

6. Document Assumptions and Limitations: Make note of any assumptions or limitations mentioned in the code or any accompanying documentation. Understand the constraints or specific conditions under which the software operates and the expected behavior in different scenarios.

7. Debug and Test: Perform debugging and testing to gain a deeper understanding of the software's behavior. This involves running the software with different inputs, observing the output, and analyzing how the code responds to different scenarios. This process helps uncover potential issues, bugs, or unexpected behavior that may provide insights into the software's inner workings.

8. Consult Resources: Consult additional resources such as documentation, user manuals, or relevant literature related to the software. This external information can provide valuable insights into the software's purpose, intended functionality, and design choices.

By following these steps, developers can gradually gain a better understanding of the existing software, its functionalities, data structures, algorithms, and limitations. This understanding serves as a foundation for further analysis, design, and improvement of the software.