SCMP/SPMP

Project Scope

The WinChangeMonitor project's scope includes designing, creating, and delivering a Windows-based monitoring tool that can track and report on file system and registry changes brought about by third-party scripts and executables. The goal of this project is to offer a workable solution that blends an intuitive user interface with the technical precision of forensic-grade monitoring.

Within the scope of this project, the team will develop a WinForms application using C# and the .NET Framework 4.8 as the core programming environment. The system will support baseline and post-execution inventories to capture registry entries, file attributes, and directory structures, followed by comparison and difference analysis to highlight modifications. Windows APIs will be leveraged to provide the necessary low-level system access, while reporting features will be designed to present results in a clear, human-readable format with options for structured export in JSON or XML. Configuration management will be maintained through GitHub to ensure effective version control and team collaboration, and the development process will be supported by CI/CD practices using Jenkins, Docker, and GitHub to automate builds, testing, and deployment. An optional SQL database may also be integrated to store system inventory data in larger-scale environments.

The project will not expand into advanced forensic analysis, integration with enterprise-scale SIEM solutions, real-time malware detection and remediation, or platform support beyond the Windows family of operating systems. These areas are considered out of scope in order to maintain focus on the project’s objectives. A Minimum Viable Product that can record inventory, analyse differences, and report will be one of the project's deliverables. Other deliverables will be exportable reports for documentation and auditing needs. The project will be completed with a final release version of the application that includes installation instructions and user documentation.

By defining these boundaries, the project ensures that WinChangeMonitor will deliver a reliable and usable tool that supports developers, system administrators, and security analysts in managing risks associated with third-party executables, while staying aligned with the available time, resources, and technical constraints.

Project Roles and Responsibilities

Project manager will be responsible for setting up GitHub for code collaboration as well as JIRA for project management and to ensure all tasks are completed before each sprint deadline. JIRA will serve as our primary tool for logging tasks in each sprint as we practice the Agile methodology.

The software developers will be responsible for writing a significant portion of the code and assigning sub-tasks to the rest of the team depending on project needs.

UX designer will work closely with the software developers to build the interface which will display the final outputs and will later be presented at the end of the project.

The quality engineer will ensure proper code practices are maintained across the project. The quality engineer in collaboration with the developers, will write the necessary tests (unit, contract, integration, etc.) or documentation to maintain code functionality.

Each team member is given a primary role in addition to contributing duties that support the primary roles of others in order to guarantee equitable and balanced work distribution. This approach avoids concentrating the workload on one person while encouraging accountability, cooperation, and cross-functional learning. The only exception is the position of Project Manager/Configuration Management, which is given to just one person in order to preserve coordination and clear leadership. Lastly, in order to ensure that every team member contributes to both the content and the review process, all reports produced as part of project management will be created collaboratively.

The table below illustrates the distribution of roles and responsibilities within the team:

	Princely Oseji	***Jeff Rose***	***Yeryoung Kim***	***Anjian Chen***	***Yu Wu***
Project Manager/Configuration Management	Primary	-	-	-	-
Software Development	Contributing	Primary	Contributing	Contributing	Primary
UX Designer	Contributing	Contributing	Primary	Contributing	Contributing
Quality Engineer	Contributing	Contributing	Contributing	Primary	Contributing

Project Management Tools

To ensure effective planning, collaboration, and delivery, this project will make use of a combination of project management, configuration management, and development tools:

- ***Scheduling (Microsoft Excel – Gantt Chart*)**:** Microsoft Excel will be used to develop and maintain a Gantt chart for project scheduling. This will provide a timeline-based view of task dependencies, milestones, and critical paths, enabling effective progress monitoring and stakeholder communication. See **appendix** below for initial project schedule.
- ***Task Management (JIRA*)**:** JIRA will serve as the primary Agile project management tool. It will be used to define user stories, assign and track story points, manage the product backlog, and monitor sprint progress. This ensures transparent task allocation and iterative planning.
- ***Configuration Management (GitHub with Git):*** GitHub, supported by Git, will be used for version control and configuration management. It will enable collaborative development, branching and merging strategies, and traceability of changes throughout the project lifecycle.
- ***Development Environment (Visual Studio):*** Visual Studio will serve as the integrated development environment (IDE) for building the application in C# and .NET Framework 4.8. It provides robust tools for coding, debugging, and testing within a unified workspace.
- ***Continuous Integration and Deployment (Docker, GitHub):*** CI/CD pipelines will be established using GitHub integrations. Docker containers may be employed to ensure environment consistency and facilitate deployment across different systems. Automated builds and testing will enhance reliability and reduce integration risks.

- ***Programming Languages (C#, SQL, JSON/XML, JavaScript):*** The core application will be implemented in C# using .NET Framework 4.8 with WinForms for the graphical interface. SQL may be employed for database storage of system inventory data, while JSON/XML will be used for structured data export. JavaScript may also be utilized for supplementary components where needed.

With its structured yet adaptable environment, this set of tools ensures robustness, maintainability, and traceability throughout the software development lifecycle. It also aligns with established project management standards and Agile delivery principles.

Risk Management Tools

Given the collaborative nature of this group project, the primary risk is the potential inability to deliver the final product by the project deadline due to differing work schedules and availability across team members. Delivering a Minimum Viable Product (MVP) that satisfies the requirements listed in the project backlog will be the project's top priority in order to mitigate this. The team will use a weighted average model or Multi-Criteria Decision Analysis (MCDA) to support a pairwise comparison of features in order to methodically evaluate and manage this risk. By using these methods to assess and prioritize features, the MVP will prioritize the most important functionality. In order to facilitate incremental development and lessen the possibility of schedule slippage, more features will be planned for inclusion in later sprints.

This structured approach provides a balance between risk mitigation and iterative value delivery, ensuring that even in the event of resource or scheduling constraints, a functional product is delivered on time with incremental enhancements planned thereafter.

Estimation and Scheduling

In software engineering practice, estimation has often been approached through the Lines of Code (LOC) method, where the anticipated size of the code base serves as an indicator of effort and project duration (as referenced in traditional models such as COCOMO). When requirements are clearly stated, LOC can be useful, but its precision greatly depends on the availability of a comprehensive specification. At this point, LOC-based estimation is not dependable because the requirements list is still being finalized.

Considering the nature of the deliverables for this project, which extend beyond coding to include activities such as analysis, validation, and documentation, LOC provides an incomplete measure of total effort. To align with Agile project management principles and the practices outlined in standards such as the ***PMBOK Guide*** and ***IEEE 1058*** (Software Project Management Plans), we will employ relative estimation using story points. Story points offer a more holistic measure by incorporating complexity, uncertainty, and the comparative scale of tasks rather than focusing solely on code volume.

JIRA will be used for work planning, tracking, and improvement. Tasks will be assigned story points to facilitate iterative planning and workload balancing. This method allows for consistent velocity tracking throughout sprints and guarantees flexibility as requirements change. The project maintains both forecasting realism and execution flexibility with this estimation strategy, both of which are essential in Agile delivery environments.

Documentation

A README file kept in the project's GitHub repository will serve as the main source of documentation for WinChangeMonitor. Anyone who needs to comprehend, install, or use the application will start with this file. It will start with a concise synopsis of the project, outlining its goals, features, and the issue it attempts to solve in contemporary Windows environments. After this introduction, the README will list the system requirements for the application, including the.NET Framework 4.8, supported Windows versions, and any extra dependencies like optional SQL database support or NuGet packages.

The README will also provide step-by-step installation instructions so that a new user can set up the project on their system without confusion. These instructions will cover cloning the repository, restoring dependencies, and building the solution in Visual Studio. Once installation is complete, the README will guide the user through running the application, performing a baseline inventory, executing a third-party program, and viewing the resulting change report. Export options, such as saving reports in JSON or XML, will also be explained.

Lastly, the README will describe how to configure certain aspects of the tool, such as specifying directories or registry keys to monitor and customizing the format of reports. It will also include contribution guidelines for developers who wish to collaborate on the project, setting expectations for branching strategies, pull requests, and coding style. Finally, it will provide links to the GitHub Issues page where bugs or feature requests can be logged, along with license information describing the terms of use.

Later in the project, if time permits, more documentation will be added, such as test case descriptions, a user manual, and a developer guide. These extended materials will only be prepared once the minimum viable product has been built and verified to function correctly. At that point, the team can focus on polishing the documentation set to support long-term maintainability and usability.

Configuration Management

Configuration management for WinChangeMonitor will ensure that all artifacts associated with the project, including source code, documentation, reports, and test assets, are consistently identified, controlled, and maintained throughout the development lifecycle. The Project Manager, who also serves as the Configuration Manager, is responsible for coordinating these efforts, while the rest of the team contributes through their assigned roles. All development will be carried out using GitHub as the central repository, with Git providing version control and GitFlow serving as the branching strategy. Semantic versioning will be followed to maintain clarity between stable releases, patches, and incremental improvements. To maintain consistency in repository structure, branch naming will follow the format CS673-(userstory), where the user story reference is drawn directly from the project backlog in JIRA.

Change requests, whether for bug fixes, enhancements, or new features, will be logged and tracked through JIRA. No changes will be introduced without prior review, and approval will be obtained through team discussions during sprint planning. Each change will be implemented through a pull request that must undergo peer review before merging into the main branch. To ensure quality and accountability, automatic merging will only be permitted once at least three team members have reviewed and approved the pull request. This process guarantees that all updates are intentional, reviewed thoroughly, and traceable. The link between JIRA tickets and Git commits will provide status accounting, ensuring that the history of modifications is transparent and easily auditable.

At the conclusion of each sprint, code reviews and baseline checks will be used to conduct routine configuration audits. A functional, tested, and documented build will be produced at the end of each sprint, providing a point of comparison for subsequent iterations. Additionally, these baselines will serve as release readiness checkpoints, guaranteeing that no unapproved or unconfirmed code is present in any system version. By confirming that tests are correctly written and run and that coding standards are upheld consistently across contributions, the quality engineer will be instrumental in maintaining code integrity.

The tools supporting configuration management include GitHub for source control, JIRA for issue tracking, and optional CI/CD integration for automated builds and testing if time permits. Together, these practices and tools establish a reliable framework for managing changes to the project, maintaining integrity across all artifacts, and providing accountability for every modification introduced. Branch naming conventions and pull request approval requirements will also be reiterated in the project's README file, ensuring that new contributors and stakeholders can quickly understand the workflow. This process ensures the project evolves in a structured, secure, and transparent manner.

Appendix

Team Contributions

Jeff Rose**:** Collaboratively, I worked with the rest of the team to complete the Project Proposal as well as the Combined SCMP/SPMP submissions in addition to discussing different approaches for how to maintain an inventory and perform a diff between the initial inventory and final system state both with and without using a designated database technology. I performed spelling and grammar checks on both the document drafts and final submissions in addition to offering a non-expert designer's eye on the formatting and layout.

Anjian Chen: We all had group meetings on September 16th and 19th, discussing SCMP/SPMP report. For our target customer, I suggested we include everyone who is using a Windows system and has concern about which files will be modified after software installation or updates, as our application is fairly general. This led into answering another question on requirements. Since we have to plan before requirements are out, I suggested we should use our group members as potential users and come up with a few requirements at a high-level for initial planning, and finalize them with details later. I also proposed that the documentation section should include some instructions for our target users to follow. Finally, I summarized what we discussed during the meetings in a goolge doc, and push it onto our group's github repository for future references.

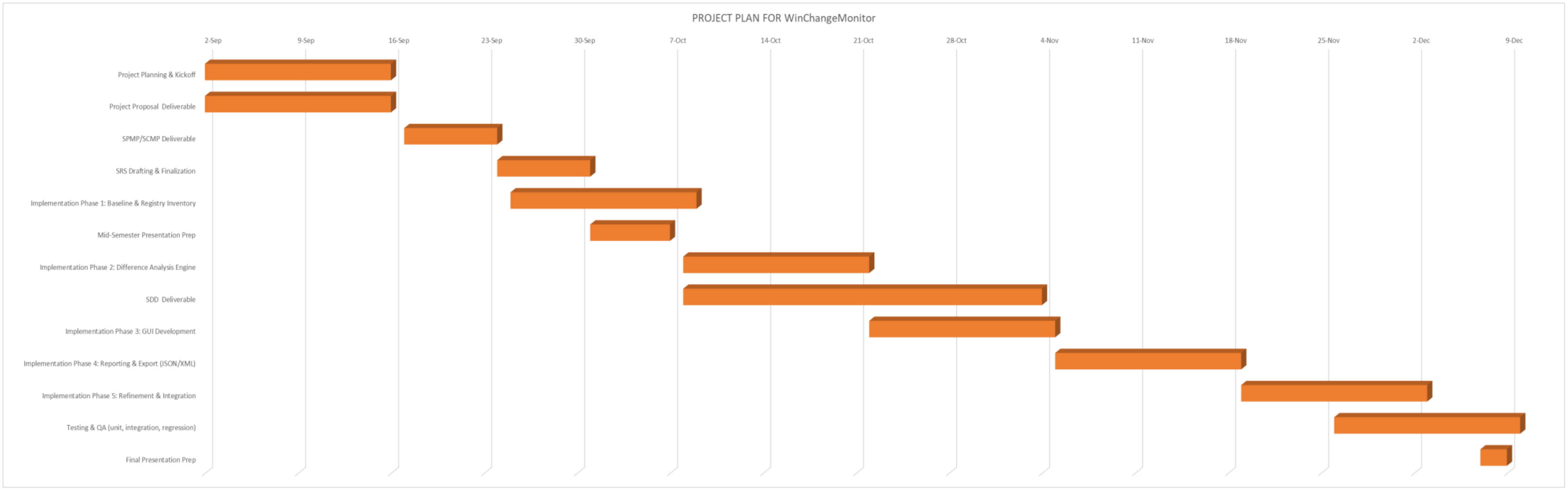
Rain Wu: On Friday, Sept 21, our team and I had a meeting to develop a Software Project Management Plan (SPMP). We went through the details of the plan, including the organization, the project management tools we will use, and how we will handle documentation and monitoring. During the meeting, I suggested that our team include several features. We decided to use story points or Lines of Code (LOC) to measure our features and to create an initial estimation of our project. Since unexpected workloads or time-consuming tasks are likely to occur during a four-month project, we all agreed that the SPMP may need to be updated. Our team members were very engaged and shared constructive suggestions. One suggestion that stood out was to implement an automated testing process. We have not yet decided how to proceed, but we will figure it out later, after we gain more understanding of the course and the project.

Yeryoung Kim: I actively participated by attending each team meeting and reviewing documents prior to submission. I also contributed ideas to strengthen our plans and documentation. For example, in the SCMP and SPMP, I suggested broadening the Risk Management section to include additional risks such as communication breakdowns, team conflicts, or reduced engagement from some members, while also recommending that we clearly outline how risks would be identified, reviewed, and monitored throughout the project. Additionally, I proposed clarifying the MVP definition to better specify which baseline features make it truly viable. In the Estimation and Scheduling section, I suggested including an initial estimate of duration and cost, even if preliminary, to make the overall plan more robust.

Princely Oseji: Collaborated with the team to create the project proposal and the combined SCMP/SPMP documents. Created the initial drafts for both documents and modified them according to feedback given by both the team and the professor.

PROJECT GITHUB: https://github.com/pco30/Software_Eng

Below is the project plan, presented as a Gantt chart, for the project.


PROJECT PLAN FOR WinChangeMonitor
2-Sep
9-Sep
16-Sep
23-Sep
30-Sep
7-Oct
14-Oct
21-Oct
28-Oct
4-Nov
11-Nov
18-Nov
25-Nov
2-Dec
9-Dec
Project Planning & Kickoff
Project Proposal Deliverable
SPMP/SCMP Deliverable
SRS Drafting & Finalization
Implementation Phase 1: Baseline & Registry Inventory
Mid-Semester Presentation Prep
Implementation Phase 2: Difference Analysis Engine
SDD Deliverable
Implementation Phase 3: GUI Development
Implementation Phase 4: Reporting & Export (JSON/XML)
Implementation Phase 5: Refinement & Integration
Testing & QA (unit, integration, regression)
Final Presentation Prep