PrivacyRadar

SCMP and SPMP 1.0

Software Configuration Management Plan (SCMP) and Software Project Management Plan (SPMP)

Authors:
David Osborne
Yashna Praveen
Madhur Deep Jain
Luke Therieau
Sujal Charak

Abstract

This document contains the Software Configuration Management Plan (SCMP) for the CS673 A1 Software Engineering course project at Boston University. It closely follows the IEEE 828-2005 Software Configuration Management standard. It establishes policies and procedures for managing configuration items and deliverables throughout the course of the project. This document also incorporates the Software Project Management Plan (SPMP) which covers team organization, project management, risk management, estimation and scheduling, and monitoring approaches.

Table of Contents

Introduction

1.1 Purpose

The purpose of this document is to establish the policies, procedures, and responsibilities needed to effectively manage the software configuration items and deliverables produced over the course of the project life cycle. This document ensures that all team members follow a standardized approach to version control, documentation, risk management, and release practices.

This SCMP exists to:

- Provide a consistent framework for managing project deliverables and configuration items.
- Ensure compliance with software engineering best practices and course requirements.
- Facilitate clear communication and collaboration strategies across the development team.
- Support quality, traceability, and timely delivery by reducing risks associated with unmanaged alterations.

The primary audience of this document includes the development team responsible for reviewing, maintaining, testing, and assessing the project. The plan laid out will serve as a reference guide for configuration management activities, tracking progress, and ensuring controlled evolution of the software system.

1.2 Scope

The Software Configuration Management Plan applies to all phases of the project, from initial requirements gathering through final delivery. It covers the management of all project-related configuration items including documents, design specification, source code, test cases, and project management deliverables.

The scope of the plan includes establishing methods for identifying, controlling, and tracking configuration items. In addition, it defines the processes for handling change requests, approvals, and implementation, by outlining version control practices and release management procedures. The plan is intended to support both the technical and organizational aspects of the project. Having consistent, traceable, and maintainable project outputs are paramount to a successful delivery of the final system.

SPMP Management

2.1 Organization

2.1.1 Team Structure

The project team is organized as a collaborative group with clearly defined roles, but without a rigid hierarchy. While the Project Manager provides coordination and oversight, decision-making is primarily consensus-driven to encourage active participation from all members.

Team	Members
Project Setup Team	Madhur Deep Jain
UI/UX Team	Luke Therieau, Sujal Charak
Backend Team	David Osborne. Madhur Deep Jain, Yashna Praveen, Luke Therieau
AI/LLM Team	Madhur Deep Jain, Sujal Charak

Collaboration is encouraged through **pair programming** and **peer code reviews** which help maintain quality and promote knowledge sharing across team members. Responsibilities for development, documentation, and testing are distributed throughout the team. This structure provides the flexibility and shared accountability needed to ensure that all deliverables are consistently reviewed from both a technical and design perspective.

2.1.2 Roles and Responsibilities

Name	Primary Role	Secondary Role
David Osborne	Subject Matter Expert	Back-end Developer
Luke Therieau	UX Designer	Data Engineer
Madhur Deep Jain	Full Stack Developer	Data Scientist
Sujal Charak	UX Designer	AI Engineer
Yashna Praveen	Subject Matter Expert	Back-end Developer

Each team member has responsibilities that align with their expertise.

- Luke Therieau, as Project Manager, oversees project coordination, facilitates team meetings, and tracks progress, while also contributing to UX design and data engineering tasks.
- Madhur Deep Jain handles full-stack development, DevOps, and data science, ensuring smooth end-to-end development, deployment pipelines, and data integration.
- Yashna Praveen focuses on back-end development with a specialization in AI integration.
- David Osborne contributes as a Back-End Developer and Network Specialist, and also serves as the Documentation Lead, maintaining accuracy and consistency of configuration items' documentation.
- Sujal Charak supports the team as a UX Designer and AI Engineer, combining user-focused design with advanced AI engineering expertise.

Configuration management is a shared responsibility across the team. Change approvals and releases are determined collaboratively to maintain accountability and quality, though the Project Manager has the final say.

2.2 Project Management Approach

2.2.1 Project Management Tools

The team uses a variety of tools to plan, track, and coordinate project activities and tasks. **Trello** is employed for backlog management, task assignment, and progress tracking, with mutually agreed-upon tasks assigned to team members and monitored for completion. **Agile methodologies** guide the team's workflow with two-week iterations, bi-weekly standups for progress synchronization and blocker identification, sprint planning sessions to define iteration scope, and sprint reviews/demos to showcase working features.

Version control is handled using Git on **GitHub**, ensuring all code changes are tracked, reviewed, and verified through pull requests. The team enforces **Conventional Commits** with standardized types including *feat* (new user-facing features), *fix* (bug fixes), *docs* (documentation changes), *style* (formatting), *refactor* (structural improvements), *perf* (performance improvements), *test* (testing updates), and *chore* (maintenance tasks, build changes, CI/CD updates). All commits must be signed and verified using GPG or SSH key signing. **Code reviews** involve at least one to two reviewers per pull request following a structured checklist: the code builds and passes tests, matches style guidelines, and documentation is updated.

Visual Studio Code serves as the primary development environment, supporting TypeScript, JavaScript, Node.js, and Electron development, with full integration with GitHub for version control. Code style is enforced using Prettier for formatting and ESLint with Typescript rules for linting. Branching follows the convention of feature/xxx, fix/xxx, and chore/xxx with all changes requiring pull request reviews before merging.

2.2.2 Documentation Tools

All documentation is written collaboratively and stored centrally on **Google Drive**, ensuring easy accessibility. Documents follow a semantic versioning approach, with major versions indicating significant updates (ex. 1.0 to 2.0) and minor versions for

smaller changes (ex. 1.0 to 1.1). Every document includes a version history and change log to ensure traceability.

Reviews and approvals are managed by the team under the coordination of the Project Manager, ensuring that content is accurate, complete, and consistent. Previous versions of documents are archived within Google Drive using the built-in 'document history' feature, allowing the team to track changes, revert if necessary, and maintain a complete record of project documentation. Additionally, coding-related documentation and other configuration items are version controlled in Git, with commit messages following the Conventional Commits standard, and formatting/linting enforced using Prettier and ESLint. This ensures consistency across technical and project documentation.

Continuous integration is performed using GitHub Actions, which runs automated processes on every push to main or pull request including unit tests via Vitest, linter and formatter checks, Electron + React application builds, and signed commit verification. Continuous delivery follows the same semantic versioning principles that were previously mentioned.

2.3 Risk Management Approach

2.3.1 Risk Management and Identification

The team has identified key project risks that could impact delivery and quality. **Technical risks** include integration challenges between AI components and network monitoring systems, potential performance issues with real-time data processing, and compatibility problems across different operating systems. In addition, **schedule risks** involve team member availability conflicts, and underestimation of feature complexity. **Quality risks** include inadequate testing coverage, configuration management issues, and documentation inconsistencies.

Each risk is assessed using quantitative analysis to determine appropriate mitigation strategies. Risk assessment considers both technical and resource constraints, with mitigation plans developed collaboratively by the relevant team members. The Project Manager maintains a list of these risks that is reviewed during sprint planning to ensure proactive risk management throughout the project lifecycle.

2.3.2 Risk Mitigation and Owners

The team prioritizes risks using a quantitative scoring method that consists of Likelihood (L), Impact (I) and Cost of Managing (M).

- Likelihood is the probability of the risk occurring
- Impact is the severity of the risk if it occurs
- Cost of Managing is the effort or resources required to mitigate the risk
- Priority is the overall risk priority, where lower P values indicate higher priority

The formula used to create risk priority is:

$$P = (11 - L) \times (11 - I) \times M$$

For example, the team has considered the risk of having to change the framework from Electron to something else if it doesn't satisfy the project's needs:

- Likelihood of changing the framework: 2
- Impact this would have on the project: 6
- Cost of mitigating this risk: 3
- $P = (11 - 2) \times (11 - 6) \times 3 = 135$

It is very unlikely that the team will change the framework. However, if it were to happen, it would have a decent impact on the project. By taking some time to research the framework beforehand, this problem can easily be avoided, making this a low priority risk. Ultimately, the formula allows the team to focus on creating back-up plans for the most critical issues should any arise.

2.4 Estimation and Scheduling

2.4.1 Project Iterations

The project follows an **Agile development** approach with **6 iterations**, each lasting **two weeks**. Each iteration is designed to deliver a minimum viable product (**MVP**) that can be demonstrated and evaluated. The team's cost estimation methodology revolves around the number of hours it would take to complete a task for effort estimation. Each user story is estimated using estimated hours (2, 4, 8, 16), and if a story is estimated to cost 16 hours or more, it will be split up into sub tasks. Resource allocation is handled as follows:

- Total team capacity per iteration: 128 hours
- Individual capacity per team member varies, but the average is around 24 hours per iteration
- Buffer allocation for unforeseen issues: 12 hours

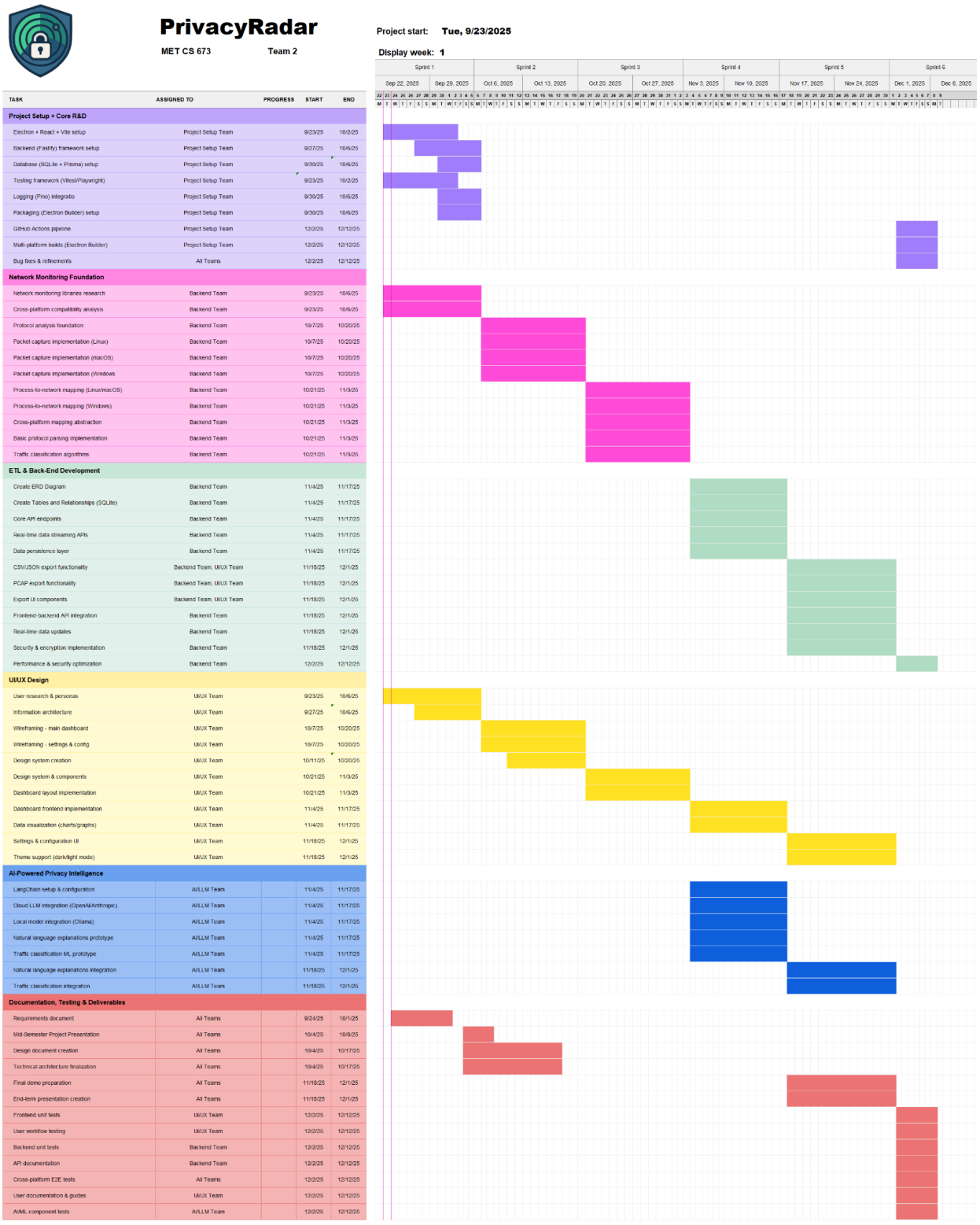
Project costs and effort are monitored through **burndown charts** in Excel and visualized during sprint reviews. The Project Manager compares actual versus estimated effort on a weekly basis, comparing story completion against planned capacity. When effort overruns are identified, the team addresses them through scope adjustment within the current iteration, delaying lower-priority features to maintain MVP delivery schedules. Cost control is maintained by enforcing the established buffer allocation and conducting check-ins to identify potential issues before they impact the team's goals.

After each iteration, the Project Manager analyzes estimation accuracy to identify patterns in over/under estimation. Refinements occur through continuous comparison of actual effort against initial estimates to improve accuracy in future iterations. The team maintains an estimation baseline that evolves based on actual performance data, ensuring that future planning becomes more precise over time.

2.4.2 Milestones, Timelines, and Deliverables

The project is organized into two-week iterations, with each iteration resulting in a minimum viable product (MVP) that demonstrates the current functionality of the software project. Key milestones include the completion of an MVP every two weeks, targeted testing phases, and the delivery of updated documentation, source code, and supporting configuration items. Milestone achievement is verified by ensuring the MVP meets all sprint checklist and task goals; functional requirements, feature completeness, and passing tests. The Project Manager is responsible for milestone sign-off by confirming the MVP satisfies project expectations.

Progress is tracked by using a **Gantt chart**, which maps tasks, dependencies, and expected completion dates from the project's inception to final delivery. Delays may arise for a myriad of reasons such as team members' personal schedules, changes in development tools, or unexpected errors within the code. To handle such situations, lower-priority features may be deferred if necessary, in order to ensure that an MVP is still delivered at the end of each sprint. This structure ensures continuous progress and aligns milestones directly with configuration deliverables.



PrivacyRadar

MET CS 673 Team 2

Project start: Tue, 9/23/2025

Display week: 1

TASK	ASSIGNED TO	PROGRESS	START	END
Project Setup + Core R&D				
Electron + React + Vite setup	Project Setup Team		9/23/25	10/2/25
Backend (Fastify) framework setup	Project Setup Team		9/27/25	10/6/25
Database (SQLite + Prisma) setup	Project Setup Team		9/30/25	10/6/25
Testing framework (Vitest/Playwright)	Project Setup Team		9/23/25	10/2/25
Logging (Pino) integratio	Project Setup Team		9/30/25	10/6/25
Packaging (Electron Builder) setup	Project Setup Team		9/30/25	10/6/25
GitHub Actions pipeline	Project Setup Team		12/2/25	12/12/25
Multi-platform builds (Electron Builder)	Project Setup Team		12/2/25	12/12/25
Bug fixes & refinements	All Teams		12/2/25	12/12/25
Network Monitoring Foundation				
Network monitoring libraries research	Backend Team		9/23/25	10/6/25
Cross-platform compatibility analysis	Backend Team		9/23/25	10/6/25
Protocol analysis foundation	Backend Team		10/7/25	10/20/25
Packet capture implementation (Linux)	Backend Team		10/7/25	10/20/25
Packet capture implementation (macOS)	Backend Team		10/7/25	10/20/25
Packet capture implementation (Windows	Backend Team		10/7/25	10/20/25
Process-to-network mapping (Linux/macOS)	Backend Team		10/21/25	11/3/25
Process-to-network mapping (Windows)	Backend Team		10/21/25	11/3/25
Cross-platform mapping abstraction	Backend Team		10/21/25	11/3/25
Basic protocol parsing implementation	Backend Team		10/21/25	11/3/25
Traffic classification algorithms	Backend Team		10/21/25	11/3/25
ETL & Back-End Development				
Create ERD Diagram	Backend Team		11/4/25	11/17/25
Create Tables and Relationships (SQLite)	Backend Team		11/4/25	11/17/25
Core API endpoints	Backend Team		11/4/25	11/17/25
Real-time data streaming APIs	Backend Team		11/4/25	11/17/25
Data persistence layer	Backend Team		11/4/25	11/17/25
CSV/JSON export functionality	Backend Team, UI/UX Team		11/18/25	12/1/25
PCAP export functionality	Backend Team, UI/UX Team		11/18/25	12/1/25
Export UI components	Backend Team, UI/UX Team		11/18/25	12/1/25
Frontend-backend API integration	Backend Team		11/18/25	12/1/25
Real-time data updates	Backend Team		11/18/25	12/1/25
Security & encryption implementation	Backend Team		11/18/25	12/1/25
Performance & security optimization	Backend Team		12/2/25	12/12/25
UI/UX Design				
User research & personas	UI/UX Team		9/23/25	10/6/25
Information architecture	UI/UX Team		9/27/25	10/6/25
Wireframing - main dashboard	UI/UX Team		10/7/25	10/20/25
Wireframing - settings & config	UI/UX Team		10/7/25	10/20/25
Design system creation	UI/UX Team		10/11/25	10/20/25
Design system & components	UI/UX Team		10/21/25	11/3/25
Dashboard layout implementation	UI/UX Team		10/21/25	11/3/25
Dashboard frontend implementation	UI/UX Team		11/4/25	11/17/25
Data visualization (charts/graphs)	UI/UX Team		11/4/25	11/17/25
Settings & configuration UI	UI/UX Team		11/18/25	12/1/25
Theme support (dark/light mode)	UI/UX Team		11/18/25	12/1/25
AI-Powered Privacy Intelligence				
LangChain setup & configuration	AI/LLM Team		11/4/25	11/17/25
Cloud LLM integration (OpenAI/Anthropic)	AI/LLM Team		11/4/25	11/17/25
Local model integration (Ollama)	AI/LLM Team		11/4/25	11/17/25
Natural language explanations prototype	AI/LLM Team		11/4/25	11/17/25
Traffic classification ML prototype	AI/LLM Team		11/4/25	11/17/25
Natural language explanations integration	AI/LLM Team		11/18/25	12/1/25
Traffic classification integration	AI/LLM Team		11/18/25	12/1/25
Documentation, Testing & Deliverables				
Requirements document	All Teams		9/24/25	10/1/25
Mid-Semester Project Presentation	All Teams		10/4/25	10/8/25
Design document creation	All Teams		10/4/25	10/17/25
Technical architecture finalization	All Teams		10/4/25	10/17/25
Final demo preparation	All Teams		11/18/25	12/1/25
End-term presentation creation	All Teams		11/18/25	12/1/25
Frontend unit tests	UI/UX Team		12/2/25	12/12/25
User workflow testing	UI/UX Team		12/2/25	12/12/25
Backend unit tests	Backend Team		12/2/25	12/12/25
API documentation	Backend Team		12/2/25	12/12/25
Cross-platform E2E tests	All Teams		12/2/25	12/12/25
User documentation & guides	UI/UX Team		12/2/25	12/12/25
AI/ML component tests	AI/LLM Team		12/2/25	12/12/25

(PrivacyRadar Gantt.xlsx is attached with submission, template created by Vertex42.com)

2.4.3 Metrics and Monitoring

The team collects and monitors several project metrics to track progress and guide decision making. Key metrics include the **completion percentage** of product backlog user stories. These metrics provide a clear view of sprint progress, task completion, and remaining workload, which helps the team adjust their priorities and resource allocation as needed.

Progress is monitored through status meetings, dashboards, and reports, with regular updates incorporated into an Excel spreadsheet for easy tracking and visualization. While the Project Manager leads the review and analysis of metrics, all team members contribute by reporting their progress and anything blocking their ability to move forward with specific tasks. Such monitoring occurs throughout each two week iteration to ensure that MVPs remain on schedule.

Software Configuration Management Activities

3.1 Configuration Identification

3.1.1 Identifying and Acquiring Configuration Items

Configuration items (CIs) are the key project deliverables that must be managed and controlled throughout the software development process. For this project, the following items are treated as CIs and are selected based on their importance to project functionality, traceability, and reproducibility:

- Requirements Documents
- Design Diagrams
- Source Code
- Test Cases and Test Results

- Library Documentation
- Build Configuration Files
- Deployment Scripts
- CI/CD Pipeline Definitions

Configuration items are managed at multiple levels including individual files, modules, and the entire repository, depending on their scope and criticality. Baselines are defined at the completion of significant milestones and/or iterations, and serve as reference points for future development. New configuration items are brought under configuration control through a review process that analyses their relevance and impact on the project. Requests for new CIs can be made by any team member, with approval handled collaboratively and final authorization by the Project Manager. Once approved, CIs are stored in appropriate repositories such as Google Drive for documentation, GitHub for code and technical configurations, Trello for project management items, or Excel for tracking and metrics.

3.1.2 Naming Conventions

All configuration items, files, branches, and releases follow standardized naming conventions to ensure consistency. Minor updates are labeled with incremented minor versions (ex. PrivacyRadar 1.1) while major updates increment the major version (ex. PrivacyRadar 2.0). These conventions are applied consistently across all team roles, supporting clear identification of versions and simplifying the tracking of changes. Enforcement of naming rules is a team agreement.

3.2 Configuration Control and Status

All changes to configuration items are managed through a change request process. Any team member can submit a change request, which includes details such as the purpose of the change, testing instructions, and relevant links, typically via GitHub pull request templates for code, or Trello cards for other items. Priority is assigned collaboratively, with changes impacting multiple areas or blocking progress taking precedence.

Change requests are evaluated by relevant team members based on criteria including code style, adherence to guidelines, passing tests, and successful CI/CD pipeline verification confirmed by the Project Manager. Approval typically occurs within 24 hours, though larger changes may require more time. Notifications are sent via GitHub and Trello to ensure all members stay informed. The status of CIs is tracked using Git and Trello, with changelogs and version histories updated whenever changes occur. Peer reviews are conducted as needed, with only relevant team members participating.

3.3 Interface and Release Management

All internal and external interfaces are documented and maintained to ensure compatibility and consistent integration. Changes to interfaces are controlled through the same change request and approval process, with the Project Manager overseeing sign-offs, and relevant team members providing technical evaluation. Interface issues are also tracked in GitHub or Trello to maintain both visibility and accountability.

Releases follow the semantic versioning as documentation, distinguishing between major, minor, and patch updates. Each release undergoes a structured process of building, testing, packaging, and deployment. When the Project Manager approves releases, all team members will be notified through GitHub and Trello. Past releases are archived in GitHub with full version history preserved for reference. This structured release management ensures consistency, quality, and clear communication amongst the team throughout the entirety of the project.

The Software Configuration Management Plan

4.1 Schedules

The SCM schedule is directly aligned with project milestones and two week iterations. Most setup activities, such as selecting libraries, defining development environments, and establishing configuration control, are planned for the first sprint to

ensure smooth workflow in following iterations. Reviews of CIs, code, and documentation are conducted **bi-weekly** to maintain alignment with MVPs and other deliverables. SCM activities are coordinated to support timely delivery, ensuring that baselines, releases, and documentation updates are completely in sync with project deadlines. Though the Project Manager maintains an overall schedule, all team members are responsible for tracking their individual tasks to ensure that personal progress aligns with the team's milestones.

4.2 Resources

All team members contribute to SCM activities, including version control, configuration management, and documentation updates. Required tools and resources include GitHub for version control, CI/CD management and integration, along with cloud storage (Google Drive) for documents and configuration items. The team's primary resource is time, which must be allocated realistically based on individual schedules and skillsets. Additional support such as training is minimal, as team members are already familiar with the required tools. The SCM infrastructure utilizes cloud platforms and GitHub for automated builds, testing, and storage, ensuring consistent access and reliability.

4.3 Plan Maintenance and SCM Across Project Phases

The SCMP must be updated and maintained to reflect current project practices, updates, and decisions. The Project Manager owns the document, but all team members are responsible for notifying the team of required updates. These updates are triggered by any changes in project scope, configuration items, or workflows. Versioning of the SCMP is managed explicitly, with a summary of changes included to track history. Updates are communicated to the team through messages, Trello, and GitHub notifications. Maintenance practices are reinforced through pre-established guidelines, including Conventional Commits, code style enforcement via Prettier and ESLint, peer review, and CI/CD processes, ensuring that all activities remain consistent, traceable, and aligned with project standards.

Configuration management practices are used throughout all phases of development. Each phase identifies which artifacts are configuration items, how changes are tracked and approved, and how baselines are established. The phases

include User Requirements, System Requirements, Architectural Design, and Detailed Design. By applying SCM consistently, the team ensures that changes are well-documented, traceable, and aligned with project objectives.

4.3.1 User Requirements Phase

During the User Requirements Phase, **requirements documents** are treated as configuration items. These documents are stored in Google Drive to track history, using the same versioning scheme that was previously mentioned. Baselines are established at the start of each iteration, capturing the complete, official set of requirements for that sprint. **Traceability** is maintained by linking requirements to design and implementation tasks. For example, pull requests should reference the requirement ID to ensure that changes in code or tests are clearly tied back to their originating user needs.

4.3.2 System Requirements Phase

System-level requirements are more **technical** in nature than user requirements, however they will be managed the same way under the best SCM practices; versioning, review, and baselining follow the established scheme. Changes to system requirements are evaluated for their impact on the project schedule and may result in additional sprint planning, design updates, or rework. Dependencies between requirements are tracked to ensure that updates in one area (ex. Requirement B) prompt review of dependent items (ex. Requirement A). Approval of system requirement changes is handled collaboratively by the Project Manager, DevOps, and relevant team members depending on the scope. Tools such as Git, Trello, and Google Drive all help support tracking and documentation.

4.3.3 Architectural Design Phase

In the Architectural Design Phase, **high-level design documents** are considered configuration items. These include architecture diagrams, module interface documents, and UML diagrams. Design versions are controlled automatically through Google Drive versioning. Design changes, such as adding a service or modifying the database schema are reviewed by the team and approved by the Project Manager.

Baselines are established to determine when the architectural design is stable enough to proceed to coding. Traceability is key to the team's success during this phase, so references to requirement IDs in diagrams are used to ensure that design decisions directly link back to user and system requirements.

4.3.4 Detailed Design Phase

During the Detailed Design Phase, low-level artifacts such as class diagrams, data schemas, APIs, and module-level documentation are treated as configuration items. These will be versioned and stored using Git and Google Drive. Inconsistencies are detected through peer reviews, testing, and automated checks when possible. The team members in charge of a specific design task will maintain these configuration items, ensuring they align with higher-level architectural decisions. Design IDs will be linked to commit messages, review checklists, and testing, signifying a clear connection from design to code and ultimately to a verified system functionality.

GitHub Link: https://github.com/madhurdeepjain/PrivacyRadar