A red square on a black background

AI-generated content may be incorrect.

Configuration Management Plan

# Version 01

2025-10-06

1000 K. L. O. Rd, Kelowna, BC

Prof. Mohamad Khajezade - Software Engineering

Volunteer Management System

**Configuration Management Plan**

**Table Of Contents:**

Contents

[Version 01 1](#_Toc210852985)

[Configuration Management System Structure 3](#_Toc210852986)

[Purpose 3](#_Toc210852987)

[Scope 3](#_Toc210852988)

[Configuration Process 3](#_Toc210852989)

[Software Versioning and Configuration Process 4](#_Toc210852990)

[Implementing Organization Documentation Configuration Process 4](#_Toc210852991)

[Responsibility & Authority 5](#_Toc210852992)

[Document and Information Types 6](#_Toc210852993)

[Software Documentation 6](#_Toc210852994)

[Software Development Process 7](#_Toc210852995)

[Software Configuration Database 7](#_Toc210852996)

[Project Management Support Information 7](#_Toc210852997)

[Workload Breakdown Structure 7](#_Toc210852998)

[Collaboration Applications 8](#_Toc210852999)

[Overview 8](#_Toc210853000)

[Git Hub 8](#_Toc210853001)

[Git Hub Workflows 8](#_Toc210853002)

[Configuration Audits and Reviews 8](#_Toc210853003)

**Configuration Management Plan**

# Configuration Management System Structure

## Purpose

The purpose of this Configuration Management Plan (CMP) is to establish processes and responsibilities for managing configurations, versions, and documentation of the **Volunteer Management System (VMS)** software project. The CMP ensures consistency, quality control, and traceability of all software artifacts developed by the Red Cross project team.

## Scope

This CMP applies to all configuration items (CIs) in the Volunteer Management System, including:

* Source code (frontend and backend)
* API definitions and schema files
* Database schema and migration scripts
* Project documentation (architecture, design, requirements, and user manuals)
* Testing scripts and datasets
* Deployment configurations and Docker files
* Continuous integration (CI/CD) workflows in GitHub

All team members are responsible for adhering to the version control and configuration standards defined herein.

## Configuration Process

The configuration process includes:

1. **Identification:** Each software component and documentation item is tagged as a Configuration Item (CI) and tracked in GitHub.
2. **Version Control:** Changes are committed to GitHub using descriptive commit messages and branch naming conventions.
3. **Change Control:** Pull requests (PRs) must be reviewed and approved by at least one peer or the Lead Developer before merging.
4. **Status Accounting:** GitHub issue tracking and milestones are used to record progress and version status.
5. **Audit and Review:** Configuration audits are conducted before major releases to verify completeness, traceability, and compliance with standards.

## Software Versioning and Configuration Process

Versioning follows **Semantic Versioning (SemVer)**:

MAJOR.MINOR.PATCH

* **MAJOR:** Incompatible API or architecture changes.
* **MINOR:** Backward-compatible feature additions.
* **PATCH:** Backward-compatible bug fixes or optimizations.

Example:

GitHub **branches** and **workflows**:

* main: stable production-ready branch.
* dev: active development branch.
* feature/<feature-name>: for new features.
* bugfix/<issue-id>: for bug corrections.
* release/<version>: pre-release staging for QA and deployment.

Merging into main requires:

* All unit and integration tests passing.
* Peer code review approval.
* Updated documentation if applicable.

## Implementing Organization Documentation Configuration Process

\*\*\*All documentation (architecture diagrams, requirements, API specs, etc.) is stored in /docs in the repository.

\*\*\*Each major revision is version-controlled and tagged.

* Documentation must be updated alongside code changes.
* Use consistent naming and date-based versioning (e.g., architecture\_v1.1\_2025-10-10.pdf).
* Architectural and design updates must be approved by the **Lead Design and Architecture** before merging.

## Responsibility & Authority

The responsibilities and authority of the members of the Red Cross project are defined by the cooperative agreement between the members themselves based on specialty in those areas. The roles of the Configuration Management Process members are outlined below.

The roles and responsibilities of the Red Cross project members are as follows:

* **Product Owner – Cade Dempsey**
* Responsible for ensuring the software meets business goals and end-user needs.
* Defines and prioritizes the product backlog.
* Approves feature scope and acceptance criteria.
* Communicates with stakeholders (professor, mock client).
* Ensures development aligns with project vision.
* **Associate Product Owner – Liam Heckrodt**
* Supports the Product Owner in requirement clarification and quality review.
* Maintains documentation and feature traceability.
* Ensures that backlog items reflect current stakeholder needs.
* Reviews system demonstrations and validates deliverables.
* Assists in defining user stories and acceptance tests.
* **Scrum Master – Kristina Cormier**
* Ensures that Agile/Scrum principles are followed and that the team remains productive.
* Facilitates sprint planning, retrospectives, and daily stand-ups.
* Removes blockers and ensures timely communication.
* Tracks sprint progress using GitHub Projects or Jira.
* Monitors adherence to the CMP and process discipline.
* **Associate Scrum Master – Hayden Nikkel**
* Assists the Scrum Master in team coordination and QA oversight.
* Supports test planning and integration validation.
* Oversees sprint documentation and retrospective summaries.
* Ensures consistent communication between development and design teams.
* **Lead Developer – Alex Anthony**
* The Lead Developer has primary responsibility for implementation and technical execution of the Volunteer Management System.
* Leads backend and frontend integration based on the architecture.
* Oversees code quality, merges, and deployment scripts.
* Conducts code reviews and performance optimization.
* Coordinates with Lead Design/Architecture to ensure technical consistency.
* Maintains CI/CD pipelines and release tagging.

# Document and Information Types

## Software Documentation

Includes: (will include)

* Requirements Specification (Volunteer Management System PDF)
* Software Architecture and Design Document
* API Reference and Data Model Documentation
* UML Diagrams (Class, Sequence, Component)
* User Manuals

All stored in GitHub /docs and versioned.

## Software Development Process

Development follows an Agile Scrum framework with 2-week sprints.  
Each sprint includes planning, implementation, code review, testing, and retrospective stages.

## Software Configuration Database

GitHub serves as the Configuration Management Database (CMDB), tracking:

* Commit history
* Pull requests and merges
* Branch status
* Milestones and issues

## Project Management Support Information

**GitHub Issues & Projects:** Task tracking, Kanban boards.

**Slack/Discord:** Team communication.

**Notion/Google Docs:** Requirements and meeting notes repository.

**CI/CD (GitHub Actions):** Continuous integration and deployment monitoring.

## Workload Breakdown Structure

Each sprint’s workload is broken down into:

**Frontend:** UI design, React component creation, API integration.

**Backend:** API endpoints, database schema, and logic.

**Testing:** Unit, integration, and system testing.

**Documentation:** Update technical and user manuals.

# Collaboration Applications

## Overview

Collaboration tools are used to coordinate communication, maintain visibility, and track changes in real time.

## Git Hub

* Source code version control.
* Branch management and merging via Pull Requests.
* Automated build and test pipelines (CI/CD).
* Wiki documentation.
* Issue tracking and milestone management.

## Git Hub Workflows

Automated workflows manage:

* Linting, building, and testing for each pull request.
* Automatic deployment to staging environments.
* Tagging releases and generating changelogs.
* Integration testing with external services (Twilio, SendGrid, Google/Outlook APIs).

Configuration Audits and Reviews

Configuration audits will occur:

* At the end of each sprint (minor review).
* Before major version releases (full audit).

Audits ensure:

* Documentation is current.
* All source code is versioned and buildable.
* Environment configurations are consistent across development, testing, and production.