**[Table of Contents]**

# Introduction

1.1 Purpose

1.2 Scope

1.3 Key Terms

1.4 References

# SCM Management

2.1 Organization

2.2 Responsibilities

2.3 Applicable policies, directives, and procedures

1. **SCM Activities**

3.1 Configuration Identification

3.2 Configuration Control

3.3 Configuration Status Accounting

3.4 Configuration Evaluations and Reviews

3.5 Interface Control

3.6 Subcontractor/ Vendor Control

3.7 Release Management and Delivery

1. **SCM Schedules**

4.1 **S**equence and coordination of SCM activities

4.2 Relationship of key SCM activities to project milestones or events, such as:

4.3 Schedule either as absolute dates, relative to SCM or project milestones or as sequence of events.

4.4 Graphical representations can be used here.

1. **SCM Resources**

5.1 Identifies environment, infrastructure, software tools, techniques, equipment, personnel, and training.

5.2 Key factors for infrastructure

5.3 Identify which tools are used in which activity.

1. **SCM Plan Maintenance**

6.1 Who is responsible for monitoring the plan?

6.2 How frequently updates are to be performed?

6.3 How changes to the Plan are to be evaluated and approved?

6.4 How changes to the Plan are to be made and communicated?

6.5 Also includes history of changes made to the plan.

# Introduction

The SepTech Centauri Software Configuration Management Plan describes the processes for documenting the software configuration management requirement approach for the Centauri application. This Software Configuration Management Plan addresses configuration management for our application. Configuration management for hardware, change management, database and development, third parties, and all other components are addressed within their own unique plan. The Software Configuration Management Plan describes the Centauri Software Configuration Management best practices. This document includes:

1. Process description
2. Process roles and responsibilities
3. Process flows and activities
4. Process scheduling
5. Process resources and
6. Process planning

# Purpose

A Configuration Item is an appointed entity for configuration management, which can consist of multiple related work products for a baseline. This grouping can provide an ease of identification and controlled access.

The purpose of Software Configuration Management, is to establish and maintain integrity for our application using:

Configuration Identification

Define the functional and physical characteristics of a Configuration Identity in sufficient detail so that it may be developed, evaluated, produced, accepted, maintained, and supported. Which can be established by baselines and approved changes.

Configuration Control

Process of evaluating, approving or disapproving, and managing changes made to items. This includes tracking each configuration for the CI’s provided, allowing new configurations to be approved if necessary, updating the baseline.

Configuration Status Accounting

The process of creating and organizing the necessary information to find the performance of configuration management. An element of configuration management that includes the recording and reporting of information needed to manage a configuration correctly.

Configuration Audit

Audits are performed to verify that an individual Configuration Item, or collection of Configuration items that create a baseline follows a specified standard or requirement. This can include functional and physical audits.

# Scope

The scope of this plan extends to configuration items that are developed or implemented for the system’s software development life cycle. This document will be used by all members of this group to provide processes, procedures, and environmental structures that will be used to verify control over application software and database components.The Configuration Management Plan is made to create integrity of the software, validity of testing, and to reduce the creation of bugs in the production environment.

# 

# Key Terms

CI - Configuration Item

# References

<https://www.guru99.com/software-configuration-management-tutorial.html#5>

<https://www.stakeholdermap.com/project-management/project-milestones.html>

# SCM Management

# OrganizationThe Configuration Management organization consists of members from the SepTech team. The project is managed under the context of scrum meetings, led by the scrum master. It is through the scrum meetings that decisions regarding configuration management activities are made. Managerial organization is primarily held responsible by the scrum master and product owner, whereas technical organization is held responsible by the development team. There is also the role of team lead, who ultimately resolves issues regarding both technical and managerial organization by making the final decision.

# Responsibilities

Configuration Management responsibilities are listed and detailed in the following table under each organizational role. For each role, their purpose and objectives, scope of authority, and operational procedures are discussed. All roles are affiliated with the SepTech team and their responsibilities are effective throughout the entirety of the project lifecycle or the duration of their role.

|  |  |
| --- | --- |
| **Roles** | **Required Responsibilities** |
| Team Lead | * Oversees the operations of the team * Provides guidance to ensure a successful sprint * Resolves issues by making the final decision |
| Scrum Master | * Ensures that the development team are on track with story tasks * Provides the development team with the necessary tools to succeed |
| Product Owner | * Tracks the progress of story tasks and product goals * Ensures that goals and tasks are sufficiently accomplished |
| Developer | * Holds responsibility for implementing story tasks decided during the initial scrum meeting of a new sprint |

# 

# Applicable Policies, Directives, and Procedures

# Currently, there are no known applicable policies, directives, or procedures that hold constraints to the development of the Configuration Management plan.

# SCM Activities

# Configuration Identification

# Identifying Configuration Items

# Activities identified for the SCM include the Centauri baseline, the system configuration items (expanding to the sublevel CIs of database, source, user interface and methods), and the configuration identification documents (expanding to specifications, plans, and procedures). These are decided by the SepTech team as items that may require the need of modifications over a timeline.

# Naming Configuration Items

The naming conventions entails unique identifiers for each item and their versions. The naming of items should be based off of the file type, configuration item title, or the combination of both. Words in the name should be separated by underscores, not spaces, and ending the with file type. If the item is a CI document and is required for a deliverable, the leading words should be the team name, followed by the deliverable and its number. The following is an example:

|  |  |
| --- | --- |
| **File Type/CI Title** | **Name Identifiers** |
| Configuration Management Plan | SepTech\_Deliverable\_4\_Configuration\_Management\_Plan.pdf |

Versions should be labeled as “vX.X” where X.X defines the version number (e.g. v1.0, v1.1, etc.).

# Acquiring Configuration Items

The configuration management library is maintained by the SepTech team. Configuration items are stored and to be accessed from the remote Github repository. This will contain all software and documentation configuration items. Access to items is given to authorized users and non-authorized users will need to request access.

# Configuration Control

# Requesting changes

Currently, changes to CIs may only be requested through the remote Github repository (https://github.com/aaronSchanck/SepTech/issues) as an issue or in the form of a pull request. To first gain access to requesting changes, contact must be made with one of the members of the development team to gain access to the repository.

# Evaluating changes

The evaluation of changes are to be done within the pull request or issue as a conversation. Additionally, changes can also be evaluated during scrum meetings. The development team is to discuss and weigh the merit of change requests in order of requests received chronologically.

# Approving or disapproving changes

Changes can be approved or disapproved either during a scrum meeting or under pull requests/issues in a conversation. In the advent of differing opinions, action taken regarding changes should be done through a voting process by the development team. Disapproved changes shall be archived and can be referenced again in the future.

# Implementing changes

Depending on the size, complexity, and urgency of the request, changes will be added to the backlog and marked as critical or non-critical. Critical changes will be developed and released in the upcoming major version, whereas non-critical changes will be rolled out in the next minor version.

# Configuration Status Accounting

# Metrics to be tracked and reported and type of report

Configuration items are to be tracked on a Kanban board. Items may be linked to Github Issues and progress is to be reported in the conversation of Github Issues. Following the release of a version, there should be a report made detailing all the changes made. Change Requests that are submitted under Issues should have its progress reported by marking its status with the following: Submitted, Reviewed, Assigned, Opened, Closed..

# Storage and access control of status data.

Storage and access of status data can be viewed and changed by authorized users, such as the SepTech team. This information can be found under the project’s Github.

# Configuration Evaluation and Reviews

Configuration evaluations and reviews are to be performed by the development team during scrum meetings. This is to ensure that configuration items have been identified accurately and created accordingly. Evaluations and reviews are to be done on the same schedule as the team’s weekly scrum meetings with the participants being the development team.

# Interface Control

Coordination of changes to configuration items with changes to interfacing items outside of the scope of the plan will be captured and detailed in the near future under the Interface Control Document. Under this document, critical interfaces will be identified and specified with requirements for the system.

# Subcontractor/Vendor Control

# Due to the fact that the application is built with the use of PostgreSQL and Flask, the development team will need to monitor changes to such services and adjust accordingly if changes interfere with the functionality of the application.

# Release Management and Delivery

The build of the application is automatically done through continuous integration as the development team pushes progress to the remote repository. The time of releases and deliveries will be decided upon by the development team.

# SCM Schedules

# Sequence and coordination of SCM activities

Coordination of SCM activities can be found in our Sprint backlog which we update weekly keeping track of the progress of the activities.

# Relationship of key SCM activities to project milestones or events, such as:

# As configuration items are identified, specific milestones will be identified in conjunction. As the project progresses, this section of the configuration management plan will be updated accordingly. Minor releases will be related to major releases in such a way that minor releases will occur concurrent to the development of major releases. Depending on the timing of minor releases, planned minor releases may be merged with major releases. The establishment of the configuration baseline is planned for when the development team has control of the application design and is ready for application testing.

# Schedule either as absolute dates, relative to SCM or project milestones or as sequence of events.

The graphical representation in Section 4.4 displays an idealized sample of the project’s release cycle and is subject to changes as the development team progresses. For this project, releases will be planned around a quarterly lifecycle which will be concurrent with project milestones. Minor releases will entail small features and/or fixes to the application. Major releases will cover new features that are major to the functionality of the application.

# Graphical Representations

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Q1** | | | **Q2** | | | **Q3** | | | **Q4** | | |
| Minor  Development | | Minor Release |  | | | | | | | | |
| Major  Development | | | | | Major Release |  | | | | | |
|  | | | | | | Minor  Development | | Minor Release |  | | |
| Major  Development | | | | | Major Release |

# SCM Resources

# Identifies environment, infrastructure, software tools, techniques, equipment, personnel, and training.

Development Tools: Android Studio, VSCode.

Database Tools: PostgreSQL, PGAdmin4, pSQL, SQLAlchemy.

Environment Tools: Android SDK API Level 23, Python version 3.8/3.9

VCS Tools: Git, GitHub

Notable Libraries(API): Flask, SQLAlchemy

Notable Libraries(App): Room, Gson, Junit, Dagger2, RXJava2, Retrofit2 + OkHTTP3, Dexter

Testing Libraries: PyTest, Junit

UI Mockup Tools: Figma

# Key factors for infrastructure:

Our development tools were chosen due to being industry standard for developing Java and Python code, respectively.

We chose PostgreSQL(includes PGAdmin4/pSQL) as our backend database due to its support for multiple types that we decided early on were going to be necessary (JSON, location, etc.).

SQLAlchemy was chosen due to its implicit ORM capabilities, that is, its ability to generate tables based on attributes in classes declared in Python code.

We chose Android API level 23 due to it being current industry standard for development. It contains the most app functionalities whilst still including a healthy amount of devices from a backporting perspective.

GitHub was chosen as our VCS tool due to its great security and due to it being contemporary industry standard for version control.

# We used tools such as Flask and SQLAlchemy to connect to our PostgreSQL backend database, and to serve as a forward facing REST API for our application. We chose Flask and SQLAlchemy due to their community packages for RESTful services and emphasis on security, allowing for decorators with OAuth and JWT.

Chosen Android libraries are discussed below.

We chose Figma as our UI Mockup tool because of its ability to have synchronized and collaborated development.

# Identify which tools are used in which activity.

-Android Studio is used to test run our application with an Android emulator and create/edit XML files which can be used for the applications UI.

-PostgreSQL is a database software that we use to hold large groups of data in an efficient manner.

-Android SDK is the software development environment used for Android Studio.

-Visual Studio is used for editing Python files and developing the Flask API.

-Figma is a UI design application that we used to create our mock ui as a template when we create our XML files in Android Studio.

-GitHub is an application where we can share files with each other and keep up with changes made to the application.

**Android Libraries:**

Room: An abstraction of a SQLite3 database, Room allows for the creation of DAO objects and possesses runtime SQL code checking

Gson: Google Json classes allows for serialization and deserialization of classes and their attributes. Can be used to send Body parameters in sync with Retrofit and OkHttp3. Also allows for storing of class data in code, thus it is highly important for passing large data.

Junit: Testing library for Java/Android code. Industry standard to use alongside CI tools such as TravisCI. Testing validation allows for easy build processes and blame to be assigned when classes don’t test correctly.

Dagger: Dependency injection library used within our app. Helps with code cleanup and reuse by building dependency mappings.

RXJava2: ReactiveX Java 2 library binds ReactiveX styles of coding seen in the JavaScript library under the same name in Android. This helps with facilitating asynchronous code, such as when one is expecting a response from the API at an unknown time in the future.

Retrofit2: A class to abstract RESTful calls to our external API. Used in tandem with RXJava2 for asynchronous calls to the API.

Dexter: Context and Permission management class, allows for simplification of the Permission granting system.

# SCM Plan Maintenance

# Who is responsible for monitoring the plan?

Scrum Master

# How frequently updates are to be performed?

Weekly

# How changes to the Plan are to be evaluated and approved?

The team leader will evaluate all changes made to the Plan and pass it on to the Product Owner to be approved.

# How changes to the Plan are to be made and communicated?

We will maintain a Kanban board that contains cards that indicate that changes to the SCM need to be made.

# Also includes history of changes made to the plan.

|  |  |  |
| --- | --- | --- |
| Version History | Name | Date |
| 1.0 | First Release | 4/11/2021 |
|  |  |  |