Prevent, Mitigate, and Recover (PMR) Insight Collective Knowledge System (PICK)

Software Configuration Management Plan

Version 2.0

02/23/2020

**Document Control**

**Approval**

The Guidance Team and the customers shall approve this document.

**Document Change Control**

|  |  |
| --- | --- |
| Initial Release: | 2.0 |
| Current Release: | 1.0 |
| Indicator of Last Page in Document: | $ |
| Date of Last Review: | 2/23/2020 |
| Date of Next Review: | 2/25/2020 |
| Target Date for Next Update: | 2/25/2020 |

**Distribution List**

This following list of people shall receive a copy of this document every time a new version of this document becomes available:

Guidance Team Members:

Dr. Ann Gates

Dr. Salamah Salamah

Dr. Steven Roach

Ms. Elsa Tai Ramirez

Customer:

Dr. Oscar Perez

Mr. Vincent Fonseca

Ms. Herandy Denise Vazquez

Mr. Baltazar Santaella

Ms. Florencia Larsen

Mr. Erick De Nava

Software Team Members:

Mr. Valente Arellano

Mr. Tarek Haddad

Mr. Joseph Warren

Mr. Mark Williams

Mr. Ivan Torres

**Change Summary**

The following table details changes made between versions of this document

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Modifier | Description |
| 1.0 | 2/5/2020 | All team members | Mark: Section 3, Section 2  Ivan: Section 4  TJ: Section 4, Section  Joseph: Section 2  Valente: Section 1, Section 3 |
| 2.0 | 2/23/2020 | All team members | Modifications based on guidance team feedback. |

**Table of Contents**

[**1. Introduction**](#_heading=h.vroi28flkonb) **1**

[1.1. References](#_heading=h.ijxu0set6roz) 1

[**2. Software Configuration Identification**](#_heading=h.ozftrqwaf7kq) **2**

[2.1. Software Configuration Item Identification](#_heading=h.dp3z83erm8y4) 2

[2.2. Software Configuration Item Organization](#_heading=h.ss0s16desbu0) 2

[**3. Software Configuration Control**](#_heading=h.170uq2v305ut) **3**

[3.1. Documentation](#_heading=h.g5ieupdkjaz) 3

[3.2. Configuration Control Board](#_heading=h.rj63z1kdawl7) 4

[3.3. Procedures](#_heading=h.xsxwuialj7cz) 5

[**4. Software Configuration Auditing**](#_heading=h.b0hbrx2dadrw) **6**

# 1. Introduction

The purpose of the Software Configuration Management (SCM) Plan is to document and brief the customers on the configuration management process the software team seeks to establish in order to meet the needs of the system in development - the Prevent, Mitigate, and Recover (PMR) Insight Collective Knowledge System (PICK). Essentially, PICK aims to provide the Lethality, Survivability and HSI directorate (LSH) with the capability of correlating and graphically representing events that have occurred in an Adversarial Assessment (AA).

The SCM Plan defines the procedures the software team will adhere to for:

* Software Configuration Identification
  + Defining and organizing Configuration Items (CIs)
* Software Configuration Control
  + Controlling modifications and releases of CIs
* Software Configuration Auditing
  + Verifying that defined processes are being followed and that delivered versions contain intended updates

The intended audience of the SCM Plan is Dr. Oscar Perez, Mr. Vincent Fonseca, Ms. Herandy Vazquez, Mr. Baltazar Santaella, Ms. Florencia Larsen, Mr. Erick De Nava and the software development team.

## 1.1. References

[1] “GitHub features: the right tools for the job,” GitHub. [Online]. Available: https://github.com/features. [Accessed: 27-Jan-2020].

[2] Hans Van Vliet, Software Engineering, Principles and Practice, 3rd edition, John Wiley & Sons, 2008. Chapter 4. [Accessed 05-February-2020]

# 2. Software Configuration Identification

The Software Configuration Identification section details all configuration items of the system that are necessary for its creation and installation.

## 2.1. Software Configuration Item Identification

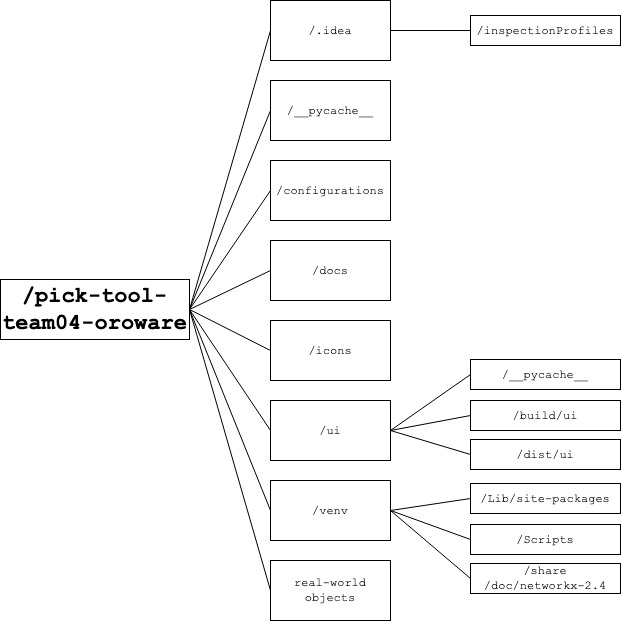
Throughout the lifecycle of software development, many iterative versions will need to be completed and reviewed. Even when a product meets all requirements and is ready for use by the client, new versions are always possible as there will always be room for improvement. Each of these versions consist of a configuration, which is the set of items needed to create and install the software. Across all of these versions, the set of items that make up the configuration may slightly change, but one thing that is certain is the modification of at least one of these items to justify a new version. The set of items that make up the configuration for the PICK tool are the following:

* Documentation
  + ***Software Requirements Specification (SRS) document:*** A document constructed by the Guidance Team that reflects the requirements of the system to be built. However, as requirements change over the development process, those changes will need to be reflected in the SRS and updated in the system.
  + ***Software Configuration Management (SCM) Plan:*** This document is the foundation for how new iterations of the system will be released, with the procedures found in section 3 affecting the pacing and organization of our modifications.
  + ***Classes Responsibilities Collaborations (CRC) document:*** This document will outline the responsibilities for each class and which responsibilities collaborate with each class.
  + ***Software Design Document (SDD):*** A software design document is a written description of a software product that a software designer writes in order to give a software development team overall guidance to the architecture of the software project.
  + ***Test Plan:*** The test plan will serve as a plan of action on how the software team will test the PICK system. This document will state the kinds of tests to run and give precise descriptions on exactly which tests are to be run.
  + ***User Guide:*** A document created by the development team for the clients with instructions for installing and maintaining the software. Likely to be created near the end of the system development.
* Source code. The item that will experience the most change over time. Each new version, whether it is a baseline or not, will have changes made to the source code.
* Test Suite. The set of all test cases we will need to test functions of the system. This is likely to evolve and grow throughout the development of the source code.

## 2.2. Software Configuration Item Organization

We, the Oroware team, will be using distributed version control with GitHub, a software product that hosts source code control tool “Git” as well as some other features [1]. Each developer will check out a full mirror of the system [2] onto their own machine and, after making necessary changes, merge with the GitHub repository to create a new version. This method of version control will allow us to have multiple copies of the system in case we lose the repository on GitHub. In the event the repository is lost, our recovery procedure will amount to uploading the most updated and complete mirror amongst our team member’s machines to a new repository. For this reason, each team member will ensure to pull the master version at least once a week to keep a backup master close at hand. It is important to note that it is highly unlikely that we will lose our source code files because the source control tool we are using, Github, is very reliable.

The initial release of the system will have a version of 1.0. Every update, which will contain at least one change, will increment the digit after the decimal point (e.g. v1.2 will be the second update after the initial release). When a new baseline is established, the digit before the decimal point will be incremented (e.g. v2.0 will be released after a major stable update has been released).



**Figure 2.1:** Visual representation of Github repository directory structure

# 3. Software Configuration Control

In this section, we describe the software configuration control mechanisms we will use in this project that will be used to control access to items in the configuration in order to prevent unauthorized updates and collisions between team members working on the system simultaneously. At a high level, software configuration control defines the process by which a new version is created in a software project. Specifically, software configuration control provides a detailed mechanism for preparing, evaluating, and approving or disapproving all change proposals to the configuration items throughout the life cycle.

## 3.1. Documentation

In this section, we review the documentation we will use to formally define a proposed change to the software system being developed. We also explain how changes to the software configuration will be documented. Essentially, we will use Github’s issues interface in conjunction with face-to-face meetings to define proposed changes. Github’s issue interface is essentially an automated change request system that has the following fields for a change request form: issue label, issue description, issue labels, issue projects, issue milestones, issue linked pull requests. A sample issue request for our team would be as follows:

|  |  |
| --- | --- |
| Issue Label: | Add Location Column to Search Page |
| Issue Description: | Based on the first demo meeting with Florencia Larsen, another column named "Location" should be added to the search criteria. The user should be able to edit, search, and sort based on "Location" |
| Issue Assignees: | tjhaddad |
| Issue Labels: | Enhancement |
| Issue Projects: | N/A |
| Issue Milestone: | N/A |
| Issue Linked Pull Requests: | N/A |

Github keeps a record of all change requests created and this will allow us to document all changes to the configuration that occur during the project, as well as analyze metrics related to the progress of the project. It is important to note that we will use Github issues to control changes to all configuration items (i.e. Github issues will be used for source code *and* documentation as well as any other pertinent artifacts.)

## 3.2. Configuration Control Board

The Configuration Control Board (CCB) will be responsible for the control and approval of changes made to the system in development, PICK. The CCB consists of all members of the Oroware software team, namely: Valente Arellano, Tarek Haddad, Joseph Warren, Mark Williams and Ivan Torres. Only the aforementioned members of the software team will have access to modify any part of the system, without restriction, provided that the correct procedures are followed when doing so. Approval or disapproval of changes will be discussed amongst the CCB and ultimately decided by considering a number of factors regarding the proposed alteration to the system.

Factors to be considered are as follows:

* Sensibility
  + Does the reasoning behind the proposed change make sense?
  + Will the change benefit the system? How?
  + Is the change based on a need?
* Required Effort
  + How long will it take for the team to implement the change?
  + Does the team have the time required to work on integrating the change?
* Scope
  + Is the change within the scope of the project?
  + Will new requirements be introduced to the project by implementing the change?
* Impact
  + Will other aspects of the project be affected by implementing the change? If so, how?
  + What are the possible risks of adopting the change?
* Priority
  + Is the change urgent? Are there more pressing tasks at hand to accomplish?

No changes should be merged into the baseline of the project nor should any changes override the work of another team member without approval being granted. In most cases, the one who proposes a change to the system will be tasked with implementing the change themselves and will need the granted approval of one other member before integration (this process is discussed in further detail in the following subsection - 3.3 Procedures). If the implementer requires assistance with building a change, they will distribute tasks to other members of the team accordingly. When a working version of the change is completed, the code is to be tested and read through by V & V who will formally and thoroughly report errors found in code. Error reports should include details of actions performed by the system that do not produce the expected result. Details on the error should be documented in the most logical and organized manner possible. Github Issues will be used to document error reports. Errors are to be labeled as “bug” when reported through Github.

The following can be considered a general outline of an error report:

1. Summary of test scenario.
2. Description of the error encountered.
3. Build or version of the system the error was found in.
4. Series of steps to reproduce the error.
5. Actual results produced by the system.
6. Expected results.

|  |  |
| --- | --- |
| Test Scenario: | Testing filter functionality on Search Logs Tab |
| Error Description: | On the Search Logs tab of the UI, a number of checkboxes (“Blue Team”, “Red Team”, “White Team”) are used to filter log entries by “Creator” or “Event Type”. Under “Creator”, when selecting the “Blue Team” checkbox, the table is incorrectly filtering by “Event Type - Blue Team”. |
| Version/Build: | v1.0 |
| Steps to Reproduce Error: | 1. Run UI 2. Navigate to “Search Logs” tab 3. Select “Blue Team” checkbox under “Creator” 4. Select “Apply Filter” |
| Actual Results: | Table is sorted by “Event Type - Blue Team” |
| Expected Results: | Table is sorted by “Creator - Blue Team” |

## 3.3. Procedures

In this section, we describe the procedures we will use to control changes to the software system being developed. Specifically, we will describe in detail what steps each team member will take to modify configuration items which include the steps necessary to define a new baseline and the approval necessary from the configuration control board throughout any given change process. It is important to note that we will be using established version control tools (essentially just Github for source code and Google Drive for all other configuration items) to aid in software configuration control for this project. We will be using, evaluating, and improving on a software configuration control system we have developed in conjunction with these tools to control changes in configuration items.

Specifically, the following steps will be taken when a configuration item must be changed:

1. During team meetings or correspondences, team members will discuss necessary changes to configuration items and team members will assign themselves tasks based on collective need.
2. A team member will create and fill out a Github issue form for the configuration item that needs to be changed.
3. The team members who were assigned the issue request will begin working on it in a sandbox environment, which in the context of our project is an experimental environment that is isolated from the production environment (e.g. branching from the master branch in Github or creating a local copy of a document)
4. The team members who were assigned the change request will complete the change request as described by the issue form.
5. The team members will be responsible for testing the completed changes they have made before integrating with the production environment. However, if a new baseline is determined after a given change, all team members will perform regression testing to ensure that the new baseline is completely functional. If any of the testing fails, the team members responsible for the change will fix their code accordingly and rerun the tests. This process will continue until all pertinent tests pass.
6. The team members who were assigned the issue request will change the status of the change request form to closed and notify other team members of the completed task.
7. The team members who were assigned the change request will integrate their changes (i.e. merge the sandbox environment with the production environment) with the baseline. If the baseline is being updated to a new version, the team members will use git tag to update the version when integrating their changes.

It is important to note that the analyst will be in charge of monitoring the issue requests as well as monitoring the software configuration control process to ensure that all team members are following the described procedures.

# 4. Software Configuration Auditing

After implementing a change in a version, we intend to ensure that the change reflects the desired requirements before releasing the new version to the client. To verify that all intended modifications and implementations were made correctly, we will use the following process:

1. Make a list of all requirements affected from the SRS
2. Go through each affected requirement and ensure the system is fulfilling the requirement
3. If the requirement is not fully met, take some time to plan out how other changes can be made to fulfill the requirement
4. Ensure all team members are following the correct processes to complete any approved changes by monitoring the change request database
5. Document any comments regarding areas of the system that may need further inspection and the overall success of the audit

It is important to note that we will also monitor configuration metrics (like average time to make a change, severity of changes, etc.) using the change request database.

**$**