MELJJ

Software Configuration Management Plan

Version 1.0

February 25, 2020

**Document Control**

**Approval**

The Guidance Team and the customer shall approve this document.

**Document Change Control**

|  |  |
| --- | --- |
| Initial Release: | 0.4 |
| Current Release: | 1.0 |
| Indicator of Last Page in Document: | $ |
| Date of Last Review: | 02/25/2020 |
| Date of Next Review: | 02/26/2020 |
| Target Date for Next Update: | 02/28/2020 |

**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. Gates

Dr. Salamah

Dr. Roach

Elsa Tai Ramirez

Jake Lasley

Customer:

Dr. Oscar Perez

Vincent Fonseca

Herandy Denisse Vazquez

Baltazar Santaella

Florencia Larsen

Erick De Nava

Software Team Members:

Jazmin Paz: Designer

Leslie Gomez: V&V

Eduardo Herrera: Systems Architect

Micheal Sansone: Lead Programmer

Jorge Flores: Systems Analyst

**Change Summary**

The following table details changes made between versions of this document

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Modifier | Description |
| 0.1 | 02/06/2020 | Jazmin Paz | Added Section 1 Introduction to SCM |
| 0.2 | 02/06/2020 | Micheal Sansone | Added Section 2.1,2.2 to SCM |
| 0.3 | 02/06/2020 | Leslie Gomez | Added Section 1.1, 3.2, 4 to SCM |
| 0.4 | 02/06/2020 | Eduardo Herrera | Added section 3.3 to SCM |
| 0.5 | 02/09/2020 | Micheal Sansone | Edited Section 2.1,2.2 |
| 0.6 | 02/09/2020 | Jorge Flores | Added Documenting (3.1) section |
| 0.7 | 02/25/20 | Micheal Sansone | Edited Sections 2.1,2.2 to now include updated procedures. |
| 0.8 | 02/25/20 | Eduardo Herrera | Edited Section 3.3 that takes feedback into consideration. |
| 0.9 | 02/25/2020 | Leslie Gomez | Edited sections 1.1 & section 4 |
| 1.0 | 02/25/2020 | Jazmin Paz | Edited section 1 |

**Table of Contents**

[**DOCUMENT CONTROL II**](#_heading=h.30j0zll)

[**Approval ii**](#_heading=h.44sinio)

[**Document Change Control ii**](#_heading=h.2jxsxqh)

[**Distribution List ii**](#_heading=h.z337ya)

[**Change Summary ii**](#_heading=h.3j2qqm3)

[**1.**](#_heading=h.3dy6vkm) **INTRODUCTION** 3

[**1.1.**](#_heading=h.1t3h5sf) **References** 3

[**2.**](#_heading=h.4d34og8) **SOFTWARE CONFIGURATION IDENTIFICATION** 4

[**2.1.**](#_heading=h.2s8eyo1) **Software Configuration Item Identification** 4

[**2.2.**](#_heading=h.17dp8vu) **Software Configuration Item Organization** 4

[**3.**](#_heading=h.3rdcrjn) **SOFTWARE CONFIGURATION CONTROL** 5

[**3.1.**](#_heading=h.26in1rg) **Documentation** 5

[**3.2.**](#_heading=h.lnxbz9) **Configuration Control Board** 5

[**3.3.**](#_heading=h.35nkun2) **Procedures** 5

[**4.**](#_heading=h.1ksv4uv) **SOFTWARE CONFIGURATION AUDITING** 6

# Introduction

The system to be developed is named the PMR Insight Collective Knowledge (PICK), and it is a tool used to perform analyses that has the capability of deriving a thorough story from log files having to do with a series of events that take place on a given network. The system is able to ingest raw log files of a multitude of formats, parse the information, and store within the system as log entries. Furthermore, the user can specify a vector to analyze, flagging significant events and introducing them as nodes of the vector. Once the vector is finalized, the system can export a visual representation of the vector, known as a graph, as well as the table representation of the vector. The purpose of the system is to facilitate and accelerate event analysis, which would otherwise take up to months to complete. The system will be dynamic in that it will allow for little user intervention, if desired, because most of the work will be performed by the system.

The software configuration management document for the PICK system serves as a means of 1. identifying configuration items (CIs) for the system, 2. controlling modifications and versions of CIs where the configuration control board that will be in charge of approving/disapproving modification to the project, and software configuration auditing to determine the degree in which the current system under development resembles the system proposed by the stakeholder.

The software configuration identification section describes the list of configuration items that comprise our system. These such CIs include the third-party Extract-Transform-Load (ETL) tool, Splunk, that our system will include, source code, user documentation, etc. Additionally, it describes the organization of the CIs, such as the syntax and naming conventions source code classes will adopt, and the location in which they will be stored. The software configuration control section describes the way in which modifications and versions will be documented and kept track of. This section will also include who will oversee modifying the different parts of the system, the means for approving/disapproving modifications, and the level of permission each user has in terms of modifying others’ work. Essentially, this section defines the mechanisms and protocols set in place to control access to CIs in order to prevent unauthorized changes and inconsistencies between team members who are working on the system at the same time. Lastly, the software configuration auditing section describes the ways in which the degree of similarity between the current system and the proposed system will be measured. This section also contains the protocols set in place to ensure delivery of the correct product with all the desired features.

The intended audience of this document are our clients, Dr. Oscar Perez, Vincent Fonseca, Herandy Denisse Vazquez, Baltazar Santaella, Florencia Larsen and Erick De Nava, as well as our guidance team composed of Dr. Gates, Dr. Salamah, Dr. Roach, Elsa Tai Ramirez and Peter Hanson.

## References

[1] Dr. Roach et al, Prevent, Mitigate, and Recover (PMR) Insight Collective Knowledge System (PICK)

Software Requirements Specification, version 1.7, 2020.

# 2. Software Configuration Identification

This section defines conventions for versioning, databases used during development, document/code naming conventions as well as collaboration procedure, and back-up procedure.

## 2.1 Software Configuration Item Identification

The following is a list of the system’s configuration items:

* Source Code
* Splunk
* Software Configuration Management Plan
* Testing files and data
* User Documentation
* Selenium
* PyQt5

## 2.2 Software Configuration Item Organization

There shall be three folders src, references, and bin. Bin will contain all needed assets such as pictures. References shall contain all documentation for the project. Finally, src will contain all source code. Inside src will be ui which will contain gui. Gui will contain all code that is associated with the graphical interface separated into different folders for each purpose. Gui currently contains graph, logview, nodeview, and popups. Each of these contains all code of the gui for each view of the program. Class names shall use UpperCamelCase and capitalized with underscores for separators while lowercase with underscores will be used for everything else. Private will have underscores before the name. Creation of new folders shall be approved and done by the code console.

Github will host the code and each member will oversee backups of the current code by pulling main at minimum once a day. Each member is also responsible for pushing their code to their github branch at minimum once a day if they have worked on it. Each issue shall have its own branch that will be merged into main when the code board approves so. Backups shall be stored on github and the user’s machine. Each teammate will work on their own machine and upload from it using git to their own repository. If there is an issue of loss of data, then the user whose data is lost shall pull from github or push from their machine to replace the missing data with one or both code consoles members supervising. If it is main, then the code console shall be responsible for replacing the data by the same means as mentioned before. Any merges or changes to main requires the members of the code console to pull the changes from github immediately

Versioning shall go as such a.b.c.d with a being major release such as the finishing of the program, b being a minor release such as adding a new gui screen or large object implementation, c shall be bug fixes, and finally d shall be minor changes such as fixing the spelling of a word in an entire file.

# 3. Software Configuration Control

In this section, we explain the process of approving or rejecting a change, how access to items will be determined, the steps that need to be followed, and how to evaluate the ramifications the change will cause

## 3.1 Documentation

To keep a better track of progress and documentation on this project, progress on the project will also be classified as a change made to this project. As a result, it will be documented accordingly throughout the semester. Progress on the project will be monitored and recorded ideally on a weekly basis with exceptions where need be. Other than that, adjustments and changes to current implementations of the project will be treated with more scrutiny, and appropriate documentation will be used.

Throughout the semester we will be working in a smaller group, and as such the process for proposing a change will not be too daunting. Propositions and suggestions can be made in group meetings or over other methods of communication, but as a group everyone will be made aware of the change being proposed before moving forward. Only after every person is made aware of a change being considered will the process of implementing or denying the change proceed. For this, we feel that meeting in person to discuss would be necessary in order to avoid any sort of misunderstanding or miscommunication. In these meetings the following will be heavily considered in the decision to either proceed or reject the change; the time and effort that would come with implementing the change and whether it can be implemented in a timely fashion, and the benefits that would come with implementation of the change and whether the rewards would outweigh the drawbacks. All this will be documented for future reference should it be necessary. Other information that will be recorded is a brief but informative description of what exactly the change is, not just the possible benefits and drawbacks. Should a change proposition be rejected, documentation on the reasons why it was rejected will be documented, also for future reference should it be necessary in the future. For a change to be accepted and implementation to begin, the decision to move forward with it must be consulted and agreed within the group. Should a proposition be accepted, documentation on why the change was accepted will be noted and implementation will be integrated into the development process.

## 3.2 Configuration Control Board

For the PICK software system team MELJJ concluded that the Lead Programmer and the System Architect will oversee evaluating and approving any changes. They will be reviewed by them and confirm if the changes shall be made. The Lead Programmer and the System Architect will be responsible for changes and approval of different parts of the source code. However, remaining members will individually contribute to different parts of the code but will be working alongside the Lead Programmer and System Architect.

The V&V will report and record any errors in code by identifying the number of bugs created and the location where the malfunction occurred. All members of team MELJJ can make suggestions to any other members code however, all changes shall be pushed into individual git branches until either Lead Programmer or System Architect approves them and merge them into master branch.

## 3.3 Procedures

We are going to be using two tools for making changes to the source code, Git and Trello. As an issue tracker, we are going to use Trello, to verify that all the changes requested are implemented in the system. Moreover, it will give us metrics on the overall performance of the team in terms of how many features are implemented in a time frame.

An *issue* is going to be added to Trello every time a client modifies or adds a new feature to the system. From there, the *issue* is going to be assigned to one of the team members to be completed by a given date (usually next demo date). From here, the team member is responsible for representing the state of the issue in Trello, for example, the feature could be in - progress, testing, or in the done state.

Once the team member is working on a feature, the team member is accountable for generating an isolated *git branch* for that feature. This allows the team members to work in parallel with the rest of the team. When an issue is completed, the team member is going to generate a pull request for merging their branch to the *dev branch*. The *dev branch* contains all the features that are going to be released for the next version (next demo). Once all the features are completed (can be verified using Trello), the team is going to integrate all their features into the *dev branch*. In this branch, the team is going to make sure that all their pieces work correctly with the rest of the components. Once proven that integration was a success, the dev branch is going to be merged with the production branch, the *master branch*. The *master branch* represents the code that is ready to display to the clients, the code in this branch is tested and verified. Once demo day comes, the system is going to be run from the production branch that contains all the features and changes the client requested.

Versions in the source code are represented with three different digits, *X.Y.Z*. The *X* digit represents the demo number, *Y* represents the feature, and *Z* represents patches in the code. For example, *3.2.0* demonstrates that we are in demo 3, have implemented the first two features, and there are no patches. The team is going to use git tags to declare when a specific feature or demo is completed. For example, if feature A is completed and tested, the feature is going to be merged into production with the corresponding version tag.

# 4. Software Configuration Auditing

The mechanism for auditing items of our project will be using a checklist to make sure everything has been captured. This audit will ensure all procedures were followed and the item satisfies the specifications that it requires properly.

* 1. Are all the anomalies being reported during a verification and validation check?

2. Are anomalies being resolved?

3. Has all source code followed the python style?

4. Can each software requirement be traced forward?

5. Have all, if any, changes been completed?

6. Have all software processes been followed and have all standards been applied?

7. Have all procedures been followed by the team?

$