

**Prevent, Mitigate, and Recover (PMR) Insight  
Collective Knowledge System (PICK) Tool  
Software Configuration Management Plan**

**Version 1.0  
February 6, 2020**

## Document Control

### Approval

The Guidance Team and the customer shall approve this document.

### Document Change Control

Initial Release:	0.1
Current Release:	0.4
Indicator of Last Page in Document:	\$
Date of Last Review:	03 February 2020
Date of Next Review:	04 February 2020
Target Date for Next Update:	04 February 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  
Elsa Tai Ramirez  
Peter Hanson  
Jake Lasley

Customer:

Dr. Oscar Perez  
Vincent Fonseca  
Herandy Denisse Vazquez  
Baltazar Santaella  
Floencia Larsen  
Erick De Nava

Software Team Members:

Eduardo A Jimenez-Todd  
Jacob N Torres  
Jorge I Felix  
Alejandro Zamora  
Matthew S Montoya

### Change Summary

The following table details changes made between versions of this document

Software Configuration Management Plan	Team6: Team404	Date 06 February 2020	Page ii
--	----------------	--------------------------	------------

Version	Date	Modifier	Description
0.1	1 February 2020	Matthew Montoya	Filled out change summary
0.2	2 February 2020	Eddy Todd	Completed SCC Section
0.3	3 February 2020	Jacob Torres	Completed SCI Section
0.4	3 February 2020	Matt Montoya	Minor UI Changes
1.0	5 February 2020	Matt Montoya	Modified Introduction, Updated References

## TABLE OF CONTENTS

<b>DOCUMENT CONTROL .....</b>	<b>II</b>
<b>APPROVAL .....</b>	<b>II</b>
<b>DOCUMENT CHANGE CONTROL .....</b>	<b>II</b>
<b>DISTRIBUTION LIST.....</b>	<b>II</b>
<b>CHANGE SUMMARY.....</b>	<b>II</b>
<b>1. INTRODUCTION .....</b>	<b>1</b>
<b>1.1. PURPOSE AND INTENDED AUDIENCE .....</b>	<b>1</b>
<b>1.2. OVERVIEW.....</b>	<b>1</b>
<b>1.3. REFERENCES .....</b>	<b>1</b>
<b>2. SOFTWARE CONFIGURATION IDENTIFICATION .....</b>	<b>2</b>
<b>2.1. SOFTWARE CONFIGURATION ITEM IDENTIFICATION .....</b>	<b>2</b>
<b>2.2. SOFTWARE CONFIGURATION ITEM ORGANIZATION .....</b>	<b>3</b>
<b>3. SOFTWARE CONFIGURATION CONTROL .....</b>	<b>5</b>
<b>3.1. DOCUMENTATION .....</b>	<b>5</b>
<b>3.2. CONFIGURATION CONTROL BOARD.....</b>	<b>6</b>
<b>3.3. PROCEDURES.....</b>	<b>6</b>
<b>4. SOFTWARE CONFIGURATION AUDITING .....</b>	<b>7</b>

# 1. Introduction

Section 1 shall introduce Team404's Software Configuration Management (SCM) document for the Spring 2020 Software II Project, *PICK Tool*, including the purpose of the document, intended audience, overview, and document references. More information on the SCM Plan is detailed below.

## 1.1. Purpose and Intended Audience

The purpose of the Software Configuration Management Document is to define and formalize the software development and configuration management process for developing the PMR Insight, Collective, Knowledge (PICK) Tool. Decisions for these processes is derived from the collaborative experiences from Team404 members, direction from the Guidance Team, and PICK Tool requirements, derived from interviews with and memos from the clients, as documented in the Software Requirements Specification Document (SRS) [1]. Although the intended audience of this document is Team404 and the Guidance Team, the clients may use and reference this document throughout the Software Development Life Cycle.

## 1.2. Overview

Section 1.2 provides an overview of the major sections of the Software Configuration Management (SCM) document. Section 1.4 is split into four subsections; one subsection for each major section of the report.

### 1.2.1 Section 1: Introduction

The SCM document begins with an introductory section, introducing the purpose and intended audience of the SCM (Section 1.1) while providing an overview the SCM document (Section 1.2). Section 1 of this document concludes by listing all external documents referenced within the text of this document in the IEEE format (Section 1.3).

### 1.2.2 Section 2: Software Configuration Identification

Section 2 of the SCM document provides a general description of the Software Configuration Identification (Section 2.1). The second half of Section 2 will define how items will be organized for Team404 (Section 2.2), including source code files and directory structures.

### 1.2.3 Section 3: Software Configuration Control

Section 3 of the SCM document includes information related to Software Configuration Control, including documentation procedures (Section 3.1), Configuration Control Board (Section 3.2), and procedures for change requests (Section 3.3).

### 1.2.4 Section 4: Software Configuration Auditing

Section 4 of the SCM document contains the process for which Team404 will use to audit project builds throughout the Spring 2020 semester.

## 1.3. References

- [1] E. Tai-Ramirez & S. Roach, SRS\_v7. Internet: <https://github.com/CS4311-spring-2020/pick-tool-team06-team-404/blob/master/doc/SRSv7.pdf>, 2020 (Jan. 30, 2020)
- [2] "Introduction to the Standard Directory Layout". Internet: <https://maven.apache.org/guides/introduction/introduction-to-the-standard-directory-layout.html>, 2020 [Feb. 5, 2020]

Software Configuration Management Plan	Team6: Team404	Date 06 February 2020	Page 1
--	----------------	--------------------------	-----------

## 2. Software Configuration Identification

This section provides labels for the baselines and their updates. The directory structure expected by Team404 to manage documents and program-code files is as follows.

### 2.1. Software Configuration Item Identification

At the top level, files describing the project meant for users to read: **README.md**. The only other files that are expected here is a **.gitignore** file, listing files and/or folders which Git should ignore and a **.git** file, containing git metadata.

There are four subdirectories of this structure: **doc**, **src**, **target**, and **test**. The **doc** directory shall contain all information material for documenting the PICK Tool project. The **src** directory shall contain all source material for building the PICK Tool project. The **target** directory shall contain all output material from building the PICK Tool project. The **test** directory shall contain all unit test code for testing software components in the PICK Tool project.

The naming conventions expected by Team404 for all source material is as follows. Naming conventions for all source material shall be short and descriptive of the contents therein. This includes the names of classes, objects, files, and variables. This convention is explored in more detail below. **Note:** All other Python naming conventions not mentioned herein shall be dictated by the python linting file that *Visual Studio Code* (VSCode) uses to perform static code analysis.

- Files and module names shall be lowercase, without underscores
  - i.e. **weblayout.py**, **trainsession.py**, **myfile.py**
- Class names shall
  - Follow the CamelCase or CapWords naming convention
    - i.e. **Node**, **Vector**, **EventConfiguration**, **MyClass**
- Method and Function names shall:
  - Use all lowercase letters
  - Use underscores ( **\_** ) to string together multiple words, as needed
    - i.e. **def generate\_vector()**
  - Use a single leading underscore ( **\_** ) in the method name to indicate the method is private
    - i.e. **def \_generate\_vector()**      **# Private Method**
- Class Methods shall have their first argument named **cls**
  - i.e.
 

```
class Hooman:
    gender = "male"
    name = "Carter"

    @classmethod
    def get_gender(cls):
        '''Access a class attribute via cls keyword'''
        return(cls.gender)
```
- Instance Methods shall have their first argument named **self**
  - i.e.
 

```
class Hooman:
    gender = "male"
    name = "Carter"

    def set_birthday(self):
        '''Create instance attribute via self keyword'''
        self.birthday = "August 8, 1998"
        print(self.name)
        print(self.birthday)
```
  - Note: The notable difference between **cls** and **self** is the method type

Software Configuration Management Plan	Team6: Team404	Date 06 February 2020	Page 2
--	----------------	--------------------------	-----------

- Variable names shall:
  - Use all lowercase letters
  - Avoid the use of single variable character names; the exception to this convention includes common language naming practices
    - i.e. **i, j** for indexes of arrays/loops/lists
  - Use underscores ( `_` ) to string together multiple words, as needed
    - i.e. **arm\_length, learning\_rate, mnist\_classifier,**
  - Use a double leading underscore ( `__` ) in the variable to indicate the variable is private
    - i.e. **\_\_secondary\_learning\_rate = 1e-4**
- Constant names shall:
  - Use all uppercase letters
  - Use underscores ( `_` ) to string together multiple words, as needed
    - i.e. **KILOMETER, ACRE\_SIZE, HOURS\_PER\_DAY**

## 2.2. Software Configuration Item Organization

The version structure expected by Team404 is as follows.

Within Team404's GitHub repository, a `ReleaseNotes.md` document shall exist within the `/doc` subdirectory. This document serves to inform the end-user what the current version number is, what fixes and or new features are included in this version, and what known issues, if any, exist. This document shall continuously be appended throughout the duration of the project such that *ReleaseNotes.md* also serves as a version history or timeline document, retaining all the historical release notes that from *version* of the project. With respect to project versions, there will be three types: Release, Feature, and Hot-Fix, with each respective build type denoted by the X.Y.Z naming convention.

The *Release* version, that is, the final version of the project which will be shipped to the clients at the conclusion of CS189b, will be denoted by the first digit in the X.Y.Z naming convention. For example, because we are shipping our first version of the project to the client at the end of the course, the version number shall be 1.0.0.

The *Feature* versions, that is, the major versions of the project which must be submitted to the guidance team and/or demoted to the clients throughout the duration of CS189b will be denoted by the second digit in the X.Y.Z naming convention. For example, because the GUI is the first major feature of PICK Tool that Team404 is tasked with creating in CS189b, the version number shall be 0.1.0.

The *Hot-Fix* versions, that is, the versions of the project which include bug fixes, performance enhancements, security fixes, and/or documentation fixes, throughout the duration of CS189b, will be denoted by the third digit in the X.Y.Z naming convention. For example, because the Team Configuration Management Plan is the first project assignment related to team-documentation, and it is the next deliverable due after the GUI (which is version 0.1.0), the version number shall be 0.1.1.

The directory structure expected by Team404 to manage documents and program-code files is as follows.

At the top level, files describing the project meant for users to read: `README.md`. The only other files that would be expected here is a `.gitignore` file, listing files and/or folders which Git should ignore and a `.git` file, containing git metadata.

Software Configuration Management Plan	Team6: Team404	Date 06 February 2020	Page 3
--	----------------	--------------------------	-----------

There are four subdirectories of this structure: `doc`, `src`, `target`, and `test`. The `doc` directory shall contain all information material for documenting the PICK Tool project. The `src` directory shall contain all source material for building the PICK Tool project. The `target` directory shall contain all output material from building the PICK Tool project. The `test` directory shall contain all unit test code for testing software components in the PICK Tool project.

The version control tool that will be used by Team404 to control source code is Git. Git is a version control system that allows Team404 to manage and keep track of the source code over time. The *Git* source code control tool will be hosted through the GitHub hosting service. GitHub provides a platform for collaborative teamwork in Team404's working environment.

The process for backing-up files, expected by Team404, for all PICK Tool project material is as follows.

All team members are to back-up all files to the GitHub repository, should any of the following criteria be met:

- At the top of the hour
- Whenever a team member must close the workstation environment for any reason.
- Whenever a team member leaves their workstation for personal reasons, including:
  - A team member leaves their workstation for a bathroom break.
  - A team member leaves their workstation for a water break.
  - A team member leaves their workstation for a snack break.
  - A team member leaves their workstation for breakfast, lunch, or dinner break.
  - A team member leaves their workstation for a walk.
  - A team member leaves their workstation when called to a different workstation.
- Whenever a team member evacuates their workstation in the event of an emergency:
  - Fires/ Fire Alarm
  - Gas/ CO2 Alarm
  - Severe Storm Evacuation
  - Shelter-in-Place /Lockdown Protocol (aka "Code Blue")
  - Active Shooter Alert
  - Nuclear Warning
  - Air Raid Warning.
  - Power Outage

Should GitHub services go offline, it is the responsibility of Team404 to back up files to the Team's Google Drive until GitHub services are back online.



### 3. Software Configuration Control

All code shall comply with the naming conventions for branching stated in this document to ease configuration control. Team members shall document their work, issues, and fixes for the System Architect to review and organize.

#### 3.1. Documentation

There shall be three types of naming conventions for GitHub branching, each for the three types of user-generated branches: Doc, Component, and Baseline branches.

Doc branches shall be branches wherein all work related to the documentation of a single feature is stored. These branches shall be branched off from a feature branch, called Baseline, following its testing phase. Feature branches are described later in this section. For example, with respect to a GUI feature, there shall be a doc branch that documents all the work as it pertains to the GUI including bug fixes, known issues, dependencies, and how the code functions. Because Team404 is documenting the project as it is being developed, doc branches serve as the culmination of all the documentation of their work for that particular feature. The naming convention for doc branches shall be <feature>-doc-v<version-number>. For example, with respect to the GUI feature, that feature shall have a branch named gui-doc-v010.

Component branches shall be branches wherein all work related to a single component of a new feature is housed. These branches shall be branched off from a feature branch, called Baseline, which is described in this section. For example, with respect to a GUI feature, there shall be a component branch for buttons, where all work for the interconnectivity between windows, is done. Because each branch contains work related to a single component, there may be many component branches for a single feature. The naming convention for component branches shall be <feature>-<component>-v<version-number>. For example, with respect to the buttons component of the GUI feature, that branch shall be named gui-buttons-v010.

Baseline branches shall be branches wherein all work, related to a new feature, is stored. These branch types branched off from the master branch, will serve as the place where the work from all components of a single feature is housed and tested. For example, the work from all components as they pertain to the GUI feature shall be merged in the baseline branch, whose naming convention for component branches shall be named gui-baseline-v010. The naming style baseline branches shall follow is <feature>-baseline-v<version-number>. When all components have been merged into the feature's baseline branch, the master branch shall also be merged into the feature's baseline branch for testing.

Within Team404's GitHub repository, KnownErrors.md and ReleaseNotes.md documents shall exist within the /doc subdirectory. Within the KnownErrors.md document, there shall be two separate sections for each classification of error: environmental and coding. Team404 considers errors related to the environment as including issues related to the OS, coding

Software Configuration Management Plan	Team6: Team404	Date 06 February 2020	Page 5
--	----------------	--------------------------	-----------

libraries, and/or computing architecture (i.e. errors specific to x86 devices or ARM-based devices) and considers errors related to coding as issues with Team404's builds of the PICK Tool, including issues related to team-written code, the UI, the functionality of the product, documentation errors, security issues, and memory leaks.

### 3.2. Configuration Control Board

All team members are expected to keep a notebook, either physical or digital, documenting their work on the PICK Tool project. In addition to ensuring team members are equally contributing to the project, this ensures that any errors and accompanying fixes/workarounds, as they relate to the project, are documented at the time of discovery.

At the end of each day, it is the responsibility of all team members to provide these documents to the System Architect, who will then organize them in the Doc branch of that particular feature or hot-fix, before a pull request is made to merge back into the master branch. Any known errors that still exist when merging branches in Git shall be included in the ReleaseNotes.md file, located within the /doc subdirectory.

Because all team members are expected to keep a notebook, they are required to document their errors, dates of discovery, the procedures for fixes/workarounds, and root causes (i.e. the root cause being the python environment, the OS, the IDE, a function call, etc.). Each section of the KnownErrors.md document shall continuously be appended by the System Architect, based on the documents provided by all team members. KnownErrors.md also serves as an error history or timeline document, retaining all the historical errors and their accompanying fixes notes, discovered while building the project.

### 3.3. Procedures

The process expected by Team404 to make a change request is through an existing pull request on the project's GitHub webpage.

To accomplish this, a user will select the pull request they wish to make a change request. After doing so, they will click on the Files changed tab and scroll down to the line(s) of code where they would like to add a comment. The user can add a comment by clicking on the blue comment (+) icon. From this field, the user can leave comments, including change requests to the existing code. When the user finishes, they click on the Start a Review or Add review comment button, followed by the Request changes and Submit review button.

Software Configuration Management Plan	Team6: Team404	Date 06 February 2020	Page 6
--	----------------	--------------------------	-----------

## 4. Software Configuration Auditing

To test that the program does what it is supposed to do we will perform unit testing which will output the test files to the /test directory. This will ease verification and validation for the processes that we intend our software system to perform.

The process expected by Team404 to be in compliance with the procedures in section 3, for all PICK Tool project material is as follows. Team404 will keep each other in line with due dates, personal accountability, and team meetings discussing the roles and progress of all project material.

\$