Keikaku 企画

PMR Insight Collective Knowledge (PICK)
Software Configuration Management Plan
Version 1.2
05/05/2020

Document Control

Approval

The Guidance Team and the customer shall approve this document.

Document Change Control

Initial Release:	0.1
Current Release:	1.2
Indicator of Last Page in Document:	[END]
Date of Last Review:	05/05/2020
Date of Next Review:	N/A
Target Date for Next Update:	N/A

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. Steve Roach, Mr. Jake Lasley

Clients:

Mr. Vincent Fonseca, Mr. Baltazar Santaella, Ms. Herandy Vazquez, and Mr. Erick De Nava

Software Team Members:

Mr. Anthony DesArmier, Mr. Angel Villalpando, Mr. Jorge Garcia, Mr. David Rayner, Mr. Valentin Becerra, Mr. Mario Delgado

Change Summary

The following table details changes made between versions of this document

Version	Date	Modifier	Description
0.1	02/03/2020	Valentin Becerra	Added Template
0.1	02/03/2020	Angel Villalpando	Completed section 1
0.2	02/04/2020	Angel Villalpando	Completed sections 1.1, 2, 2.1, 2.2
0.2	02/04/2020	Valentin Becerra	Completed section 3.3
0.3	02/05/2020	David Rayner	Completed sections 3.1 and 3.2
0.3	02/05/2020	Mario Delgado	Completed section 4
1.0	02/05/2020	Anthony DesArmier	Corrected formatting, grammar, updated section 3.3.
1.0	02/05/2020	Valentin Becerra	Reviewed Document, and fixed grammar for 3.1 and 3.3
1.1	02/13/2020	Angel Villalpando	Removed configuration items, Added more detailed directory structure, clarified configuration item organization, added more details to procedure
1.2	05/05/2020	Jorge Garcia	Updated Software Configuration Item Organization section
1.2	5/05/2020	Anthony DesArmier	Updated Software Configuration Item Organization section

TABLE OF CONTENTS

1. Introduction

The Prevent, Mitigate, and Recover (PMR) Insight Collective Knowledge (PICK) system is intended to ease the process currently used by the client in order to analyze the data generated during Adversarial Assessments (AA). Their process currently takes months to analyze the generated data and this system is meant to reduce that time by combining several elements currently used into a single system.

The Software Configuration Management (SCM) plan for this project is designed to systematically control changes to the configuration of the system at any, discrete point in time. In addition, it helps to maintain the integrity and traceability of the configuration of the system throughout its development lifecycle [2]. The SCM consists of four elements: Software Configuration Identification, Software Configuration Control, and Software Configuration Auditing.

This document consists of this introduction, the Software Configuration Identification, the Software Configuration Control, and the Software Configuration Auditing being used for this proposed SCM. The intended audience of this document is meant to be the client, as it can be used to measure the progress during the initial development and continued maintenance of the system.

1.1. References

[1] Hans Van Vliet, Software Engineering, Principles and Practice, 3rd edition, John Wiley & Sons, 2008. Chapter 4.

[2] H. E. Bersoff, Elements of Software Configuration Management. IEEE Transactions on Software Engineering, 10(1):79-87, Jan 1984.

2. Software Configuration Identification

This section is intended to identify the baselines of this SCM, which are system versions, or releases that have been formally reviewed and agreed upon.

2.1. Software Configuration Item Identification

A typical configuration of the system outlined in this SCM will consist of certain items that must be provided and proved to be modified at the time of any given release. Such items will include the source code itself, the evolving requirements documents, test suites for every iteration, and a user guide included at the final delivery of the product.

Python 3.4+, preferably 3.8 is required. This application supports any environment that has Python 3 support.

The following is a list of the current required python package installations:

- PyQt5==5.14.2
- PyQt5-sip==12.7.2
- python-dateutil==2.8.1
- python-dotenv==0.13.0
- splunk-sdk==1.6.12
- virtualenv==20.0.20
- virtualenv-clone==0.5.4

Splunk Enterprise is required in order to run the Splunk server. This requires a Splunk Enterprise download on the respective OS. The host's username, password port of the Splunk server, and index name to store the entries is required.

Storage is to be managed through serialization and saved on a file system. Therefore, currently there is no need for installation of a database.

2.2. Software Configuration Item Organization

The labeling scheme that our software team will adopt will be that our initial baseline will take on the number of v.0. Thereafter, every major feature addition will increment the tenths decimal in the baseline name. Any smaller additions in a given session to the feature would add another decimal point with a new increasing decimal number. The final deliverable program should have the version title of v.1.0.

For example, at inception, the software will take on the name v.0. When a major component or feature, such as a user interface, has been added, this shall result in the name v.0.1. If a team member checks out that particular version and adds multiple new elements to the major feature .1, say, the user interface, then the version name will take on the name v.0.1.1. The additional decimal point and new number represents the number of GitHub pushes contributing to the improvement of the major feature.

For instance, the version v.0.1.2 would represent that we are working on baseline v.0, have already added major component .1, such as the user interface, and have had .2 pushes with multiple elements that compose the user interface. This should allow us to keep track of major features and the pushes with the smaller components that compose them with a single glance at the name. The aforementioned organization of versions will not only apply to the source code of the project, but also to other documents and deliverables. They will remain independent of each other, which means that a given source code file's name and a particular document or deliverable's name may have different names and versions.

The directory structure that we will be using will be simple and can be represented in the following, hierarchical manner:

- **doc/ -** All client documents (SRS, etc.) and team documents (SCM, user-guide, etc.)
- **src/**
  - **model/ -** All .py files that model the real-world objects in the system (Log Entry, Vector, Graph, etc.)
  - **gui/ -** All .py files created by .ui references for graphical the user interface.
    - **graph/ -** All graphing tools and graphing .py files
      - **qss/ -** Style sheets used for definitions for QT graph elements
  - **util/ -** Holds utility .py files for help routines (queue and file_util)
- **ui/ -** All QML reference files.
  - **icons/ -** All icon files in .jpg, .png, or other image files used for QML icon objects.
- **tests/ -** All source code files and documentation for test suites.
- **pickData/**
  - **copies/ -** Cleansed adversarial assessment data storage.
    - **red/ -** Red team file storage.
    - **blue/ -** Blue team file storage.
    - **white/ -** White team file storage.
  - **root/ -** Assessorial assessment data storage.
    - **red/ -** Red team file storage.
    - **blue/ -** Blue team file storage.
    - **white/ -** White team file storage

Our project files will be stored on the GitHub repository created by the Guidance Team. All Keikaku team members will be working on their individual machines and will perform daily commits and pushes to the group repository. If a program source code file is changed, as opposed to changing the name of the file itself to reflect the changes, a comment block at the beginning of every file will be maintained to reflect the current version of the file. This is to keep program file names consistent throughout the development of the system and to avoid causing issues within the source code due to the natural coupling of the classes.

We will be using GitHub's automatic backup features to backup all program files and to assist in restoring to previous iterations if any team members' local machines crash or any program data becomes compromised. The team's Lead Programmer and V&V Supervisor will be the responsible parties to ensure that back-ups are taking place at least once a week, but preferably at the end of every workday. Ultimately, they will be responsible for ensuring that all program files remain recoverable throughout the system's development lifecycle.

3. Software Configuration Control

This section of the SCM is intended to outline the configuration control techniques that are to be followed in order to avoid team conflicts. The section includes the change request process and the members of the control board along with their responsibilities. Finally, it highlights the workflow in order to coordinate and manage configuration items.

3.1. Documentation

The form of documentation that we look to supplement the project is a part of GitHub's project management tools called the Project Board. On this tab, we will manage potential change requests through the "Issues" thread. This will allow each team member to post and monitor requested changes or bugs that need to be resolved before the production of the next major deliverable. In the event the client requests a necessary change or we, the development team, believe that a change is necessary, a new issue will be created. Each issue created by the development team is to include an informative title and description regarding the purpose of the change request along with a color label that will function as an indicator. An issue is to be marked with a priority level coloring scheme either high or moderate. The scheme is as follows: red labels (mandatory) for destructive bugs that require immediate attention and a blue label for change requests that are deemed necessary but do not compromise the software. Once an issue or change request is evaluated and approved by the Lead programmer and V&V Supervisor, it will then be assigned to development members that are responsible for that change. In addition to being assigned a priority, a date of completion will be decided upon in order to track milestone additions to the development of the program. GitHub will also keep track of the history of each created and completed issue, thus, creating a record or reference for future review.

3.2. Configuration Control Board

The list below identifies the Configuration Control Board members along with their roles and responsibilities throughout the software's development lifecycle.

V&V and Lead Programmer: The two will share the following joint responsibilities:

- The calling of mandatory meetings and review sessions before the release of a major deliverable. Each mandatory meeting will highlight the errors or bugs that need to be resolved, the changes in question, and the progress of such items.
- The ensuring that changes are either accepted or rejected. This acceptance criterion is evaluated based on three main factors: priority, amount of resources/time required, and the impact of the change on the software i.e. source files and dependencies.
- The management of the baseline branch via pull requests before finalized changes make their way into the software.
- The creation of backups and copies after a change has made its way into the software.

System Architect and Designer: The two will share the following joint responsibilities:

- The creation and management of issues or change requests regarding bugs.
- The management of the hotfix or bug branches via pull requests.
- The structuring of files and directories in an organized manner.

System Analyst: The analyst will have the following sole responsibilities:

- The creation and management of issues or change requests regarding enhancements either created by the development team or requested by the clients.
- The management of the hotfix or bug branches via pull requests.

Once each member is assigned a task, changes shall be accepted through pull requests by the manager of the branch. If members are in a meeting to work on a task and the branch manager(s) is/are physically in attendance and give their approval to push a change, then only at that point can a member push a task without a pull request.

3.3. Procedures

In order to determine what changes need to take place in the source code, our team will adhere to the following procedure:

1. During our team's face-to-face to meetings, we will collectively determine what are the items that need to be added, updated, or modified.
2. Once the items that need the most attention have been determined, the work will be delegated to individual team members.
3. The V&V Supervisor and Lead Programmers will be in charge of ensuring that there are no conflicts between one file that has been checked out by multiple team-members by regulating the pushes and pulls to the develop branch.

In order to control the changes that are collectively determined, we will use Git as our distributed version-control. The workflow we have chosen for this project is commonly known as Gitflow. This workflow uses a master branch, a complementary development branch, a release branch, and any number of feature branches and hotfix branches. The master branch contains only an official release history while the development branch serves as an integration branch for new features and contains a full history of the project. New features are forked from the development branch and are merged back into it when the features are completed. The Lead Programmer and V&V Supervisor will be responsible for approving the merges back into the development branch.

Once the development branch is ready for release, a release branch is forked from it. The release branch is where further documentation, bug fixes, and release-oriented tasks should be performed. When on this branch, the V&V Supervisor and Lead Programmer make sure that the software is up to specification for the release. Upon completion, the release branch is merged onto the master branch and is tagged with a version number. Hotfix branches function similarly to feature branches but are based directly off the master branch and are the only branch to fork from master. Once the hotfix is complete and been reviewed by the V&V Supervisor and Lead Programmer, it shall be merged directly into the master and develop (or current release) branches, and master tagged with an updated version number.

4. Software Configuration Auditing

This section is meant to provide the methods the development team is to follow in order to ensure the correct procedures have been established in completing software deliverables.

To ensure that a development branch has all the intended features before merging it into the master branch, weekly reviews will be performed on the current build of the system. These reviews will be conducted by our V&V Supervisor: auditing every member's work and making sure every feature implemented is correct and complies with the requirements. During a review, the commit history on each branch shall be referenced in order to analyze member habits in the workflow. An audit checklist is to be referenced during each review. The checklist is described below:

Software Configuration Audit Development Team Checklist:

- Collectively determined what items need updating/changing
- Followed git workflow procedure
- Used the issues thread appropriately
- Called upon pull requests for change reviews
- Followed file naming conventions
- Used the correct version labeling scheme
- Used descriptive commits
- Abided by the control board responsibilities
- Changes and bug fixes were consistently monitored and accepted/rejected each time
- Baseline changes were agreed upon
- Milestones were met each time

[END]