PICK

Software Configuration Management Plan

Version 1.5

2/5/20

Document Control

Approval

The Guidance Team and the customer shall approve this document.

Document Change Control

|  |  |
| --- | --- |
| Initial Release: | 1.1 |
| Current Release: | 2.0 |
| Indicator of Last Page in Document: | $$ |
| Date of Last Review: | 2/23/20 |
| Date of Next Review: | 2/24/20 |
| Target Date for Next Update: | 2/24/20 |

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. Roach, Jake Lasley

Customer: Dr. Oscar Perez, Vincent Fonseca, Herandy Denisse Vazquez, Baltazar Santaella, Florencia Larsen, Erick De Nava

Software Team Members: Ian Hudson, Diego Garcia, Wan Koo, Ricardo Pineda, Jesus Ramos Favela

Change Summary

The following table details changes made between versions of this document

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Modifier | Description |
| 1.1 | 2/4/20 | Wan Koo | After our meeting in discussion wrote Section 1.1 and 1.2 |
| 1.2 | 2/4/20 | Jesus Ramos Favela | After our meeting in discussion wrote Section 2.1 and 2.2 |
| 1.3 | 2/4/20 | Diego Garcia | After our meeting in discussion wrote Section 3.1 and 3.2 |
| 1.4 | 2/5/20 | Ricardo Pineda | After our meeting in discussion wrote Section 3.3 and 4.1 |
| 1.5 | 2/5/20 | Ian Hudson | After our meeting in discussion revised all sections and made sure formatting and documentation was consistent |
| 1.6 | 2/23/20 | Wan Koo | Refactor of sections 1 and 2.1 |
| 1.7 |  | Ian Hudson | Fixed sections 3.1 based off instructor feedback |
| 1.8 |  | Ricardo Pineda | Worked on section 3.2 |
| 1.9 |  | Diego Garcia | Refactor of sections 3.3 |
| 2.0 |  | Jesus Ramos Favela | Corrected section 4 based off instructor feedback |

Table of Contents

[Document Control ii](#_Toc61860422)

[Approval ii](#_Toc61860423)

[Document Change Control ii](#_Toc61860424)

[Distribution List ii](#_Toc61860425)

[Change Summary ii](#_Toc61860426)

[1. Introduction 1](#_Toc61860427)

[1.1. References 1](#_Toc61860428)

[2. Software Configuration Identification 1](#_Toc61860429)

[2.1. Software Configuration Item Identification 1](#_Toc61860430)

[2.2. Software Configuration Item Organization 1](#_Toc61860431)

[3. Software Configuration Control 1](#_Toc61860432)

[3.1. Documentation 1](#_Toc61860433)

[3.2. Configuration Control Board 2](#_Toc61860434)

[3.3. Procedures 2](#_Toc61860435)

[4. Software Configuration Auditing 2](#_Toc61860436)

# Introduction

The purpose of the Software Configuration Management Plan is to define and regulate the delivery of the Prevent, Mitigate, and Recover (PMR) Insight Collective Knowledge System (PICK) and to ensure that all builds and releases produced are standardized and consistent. In order to accomplish this goal the software this document defines the software configuration identification, control and auditing procedures in detail. In Section 2, Software Configuration Identification, all configuration items are clearly identified with their organizational structure to ensure clarity regarding what elements exist within each build and their intended use. This includes the directory structure at which all documents and content regarding each build can be found as well as the naming conventions used to monitor the status and version numbers of the configuration items. In Section 3, Software Configuration Control, gives detailed instructions regarding the procedures and processes in place to handle changes and updates to configuration components. This covers all material in reference to individual and team changes as well as how to mitigate conflicts should they occur between team members working on the same portion of the project at once. Lastly Section 4, Software Configuration Auditing, discusses the details regarding the way that the changes, as discussed in section 3, will be verified and validated against the needs of the clients. This includes a detailed process as to what steps are taken to ensure compliance and success of the product.

## References

1. PICK SRS\_v7

# Software Configuration Identification

This section provides labels for the baselines and their updates.

## Software Configuration Item Identification

Each configuration delivered to the client shall consist of the follow items:

|  |  |
| --- | --- |
| Item Name | Description |
| Source Code | Product’s source code necessary to compile and run the project, labeled by the ProjectName\_VersionNumber |
| Software Requirement Specification | The most recently approved requirement specification as agreed upon by the client and the leadership team noted by SRS\_VersionNumber |
| All Design Docs | All documents noting system and architecture design in detail, noted by DesignDocName\_VersionNumber |
| User Guide | The User Guide used to discuss intended features and outline usage of the product from the outside user perspective as noted by UserGuide\_VersionNumber |
| Test Suites | A Folder containing all test suite divided by name and if necessary, module, noted at the root level as TestSuites\_VersionNumber and within that folder as TestName\_VersionNumber |
| Project Plans | All Project Plans used to identify future goals or general knowledge on the project as noted by ProjectPlan\_VersionNumber |

## Software Configuration Item Organization

The labeling scheme used to measure and maintain consist version control is as follows: All baselines are defined by the whole number to the left of the decimal place with each number to the right of the decimal place defining the update number since that baseline. What constitutes an update will include relevant bug fixes, addition of needed features, and removal of unneeded code/features. For each baseline’s initial release, the update number will start with 0. All documents not including source code will be kept at the root directory of the configuration to provide easy access to necessary documentation. This includes the relevant design docs, a folder containing all test suites separated by type and then if relevant by module, Software Requirements Specification, Module Descriptions, User Guides, and additional documentation as needed. There will also be a folder containing the source code identified by the project name with the implementation of git-label (addition/removal of features will be marked), inside of which will contain a folder titled src, which contains all relevant source code necessary to compile and run the product. Within the src folder each module will be separated into their own folders used to identify and decouple modules and their resources. Alongside which there will persist a resource/asset folder and any necessary build components necessary for the project to build successfully as deemed by the libraries used within the project. All of these details will be covered in the general user guide to the project’s codebase as well as within the design docs and project plan. This project will be hosted on GitHub Classroom using the Version Control Software, or VCS, git. As we are using git, the branch structure will be as follows. All configurations will be defined based off the master branch should there be a need we can adjust this for every update release as well. For every module that is being worked on, a new branch will be created for that module. For each feature that will be worked on by an individual they will create a branch from the module branch in development named IndividualsName/Feature. All branches will be have masters changes pushed into it once per week to prevent Master from getting too behind any module branch. Backups will happen on a weekly basis at the team meeting, during the same time as weekly standup, and will rotate team members machines. This insures that even if a team member’s hardware would return inoperable or the repository for the project were to corrupt the maximum amount of work lost would not exceed one week. Recovery procedures would include creating a new repository and rebasing the master branch to the most recent backup as kept by the team. Should there be a need or high amount of concern, weekly meetings could include all members backing up the most recent configuration and source code to reduce the probability of lost work.

# Software Configuration Control

This section covers the procedures and documentation for approving and providing changes to items within the configuration as discussed in Section 2. This section will cover collision avoidance and the measures taken to prevent unapproved content from being pushed into the configuration. It will also cover the body tasked with evaluating, approving, and denying each change proposal as well as the responsibilities of each individual and the procedure for this process as well. Our Team intends to use git and its procedures as a VCS to avoid collisions and maximize collaboration

## Documentation

The documentation our team will be using to process changes will be initially a team meeting to determine what work needs to be done, followed up by a verbal notice that an individual wants to do this work most likely in the form of a task breakdown. This then will lead to the proposed changes being done and then a Pull Request being submitted and a code review being performed. that changes need to be made via one of the communication channels within the team followed by a Pull Request on the branch that the changes are to be merged too. This Verbal Pull Request, or PR, must contain the relevant features worked on by the individual proposing the change as well as a brief explanation of the implementation processes and design decisions made during implementation. It must also contain all interactions with other modules and subsystems that could potentially lead to conflicts or errors. All changes must be approved during a pull request

## Configuration Control Board

The configuration control board for this project will consist of the team members from Team 2. In order for a PR to be approved it must undergo a code review consisting of at minimum 1 other team member and the V&V. In this process the individual attempting to merge their changes will explain their process and logic while implementing their change and all members present of the Board must agree to merge the PR otherwise the PR will be denied. The PR will be evaluated based on the following criteria. It must serve its intended purpose without imposing additional computation that is deemed unnecessary. It cannot serve to implement gold plating outside of the requested scope of the needs and requirements. It cannot break any other components or processes within the module its being implemented in or outside, which is checked via tests that we will write for each of the modules. The V&V will make note of every location where an error occurs and summate the reasoning as to its failure briefly and deny the merge until all errors and broken code to be correct. Should the changes pass all criteria the changes will be merged into the branch and the module version will be updated. If it fails however the code will be denied and therefor the individual must repeat the code submission/review process.

## Procedures

The Procedure used in order to push changes within the system as briefly described above in sections 3.1 and 3.2 is as follows. Meetings will happen once a week one at the beginning of the week to determine which group member will work on for that week and the past work that they had completed the previous. At any time during the week if a group member has finished their section they then schedule time with at minimum 1 team member and the V&V. Once this is done they must submit a Pull Request to the appropriate Module branch citing the team members they wish to meet with as reviewers of their PR. The team members will then meet with the individual and perform a code review to ensure that the changes being made are consistent to the goal of their intended work as well as meets the criteria listed above in section 3.2. In the event the changes pass the code review they would then be allowed to merge their changes into the module branch they made the PR against and the module version would then update. Should the changes fail the code review the V&V would make note of each location where the changes failed the evaluation criteria and summate these issues into a reinforcement action report (report detailing why the code review did not meet the evaluation criteria set by the group) that can be returned to the individual submitting the PR in an effort to allow for adjustments and changes and eventually a resubmission of the changes. In the extreme case that the changes are entirely away from the intended focus and goal of the feature being implemented the changes may be discarded entirely and a meeting may occur in order to re-discuss the goal oriented around that feature. This process and procedure can be implemented entirely using the Pull Request system in git and GitHub. Revisions to this process and in turn document may occur as needed but will always be reported.

# Software Configuration Auditing

The mechanism we will use to make sure that the process stated in section 3 is consistent with the needs met by the client is to hold regular team meetings at minimum once a week to review all feature work and needed work. During this time, we will not only break up tasks and responsibilities, backup the project, and discuss PR’s but we will also make sure that all intended feature work is aligned directly to needs and requirements stated in the SRS. This way no work is started without the verification that it is directly aligned with needs from the client that are backed using requirements from the SRS. If the team cannot verify the feature or piece of work needed to be done based on these criteria it will not be approved to be worked on and any PR’s submitted with this feature will be immediately dismissed.

$$