IEEE Configuration Management Version (1.0)

Team Members:

Karthik Raja

Zachary McPherson

Ricky Zhao

Manelin John Raja

Thomas Eckrich

Jiahao Li

Original Issue Date: 12/12/2024

Document Number: PAAC-01

Version Number	Author	Date	Reviewed By	Signature
1.0	Karthik Raja	12/11/2024	Ricky Zhao	
1.0	Zachary McPherson	12/11/2024	Karthik Raja	
1.0	Ricky Zhao	12/11/2024	Zachary McPherson	
1.0	Thomas Eckrich	12/11/2024	Jiahao Li	
1.0	Jiahao Li	12/11/2024	Manelin John Rajan	
1.0	Manelin John Rajan	12/11/2024	Thomas Eckrich	

Configuration Management 1.0

Final Approval

Name	Date	Signature
Dr. Joseph Profeta	**12/12/2024**	
Mr. Anuj Lele	**12/12/2024**	
Mr. Thomas Gallagher-Teske	**12/12/2024**	

This document represents the current configuration management requirements (Version 1.0) for the system. As is typical in the development process, the requirements are not fully complete in the initial version. The Configuration Analysis phase that follows the CMS (Configuration Management Specification) will assess the consistency, completeness, and feasibility of the requirements. Additionally, during the CA phase, evaluations of third-party tools will be conducted to determine their compatibility and usability. After the CA phase, the CMS will be updated to reflect any necessary changes.

Table of Contents

1. Definitions and Abbreviations

1.1 Repository

Centralized places where all the files, documentation, and revision history for a project are stored.

1.2 Branch

Separate lines of development that can be created, merged, or deleted to manage different features or versions.

1.3 Commit

Saved changes to the code base, including a description of what was changed and why.

1.4 Pull Request

Proposals to merge changes from one branch into another, which can be reviewed and discussed by team members before being integrated.

2. Managing Code Base with GitHub

In our project, we utilized GitHub as a powerful tool for version control and development collaboration. GitHub allowed out team to work on the same project at the same time with the use of its many different features. One of the features that helped us maintain high standards of code quality was each of our group members having different branches within the same repository. Each member of the team having their own branch is essential for efficient software development since it allows the developers to work on their assigned portion of the project and it helps manage different features or versions. Once the developer is ready to insert their changes from their local machine to GitHub, they would have to commit their changes from their IDE. Our team committed our local changes to GitHub either every day or every time there is a bug fixed. This approach

ensured that our code base was always up-to-date and helped us maintain a smooth and efficient workflow. Another practice that we followed was making a branch for each of the iteration we had throughout the development of the project. To this day, we have not merged anything to the main branch. We have a branch named “iteratoin2-branch” and “iteration3-branch,” which has all of our code for the respective iterations. But, for our last iteration, we will be merging our changes to the main branch, since we will not have any iterations after the last iteration. Making different branches for different iterations has helped us keep track of our progress for each iteration and organize the development process. This branching strategy has also made testing and debugging easier making sure that each iteration’s changes are properly isolated before final integration. We used the Pull Request feature in GitHub in order to merge our code from our personal branches into the corresponding iteration branch. The team member that wanted to merge the changes into the iteration branch would create the Pull Request, but only another team member would review that Pull Request and confirm the request. It is not good practice to create a Pull Request and confirm it yourself. To avoid any merge conflicts in the Pull Request, the team would pull the changes from the iteration branch into their own personal branch. This would ensure that no merge conflicts would be present in the creation of the Pull Request. Furthermore, our team participated in weekly code reviews, where all of the team members would get together and discuss any issues or defects that were present in their code. The team mainly communicated with the module that they are integrated with, since those were the modules that needed to know any code changes for that specific module. But the team did have collaborative code reviews as well where all modules participated. Overall, these practices helped us develop software efficiently, keep our code quality high, communicate effectively, and finish our project successfully.

3. Tracking Defects Using GitHub

During the development of our project, we used GitHub's tools for tracking and managing defects, ensuring issues are documented, assigned, and resolved efficiently. Our team started to track defects and issues in the final four weeks of development as instructed. One of the features that helped us maintain high standards of code quality while resolving issues was the Issues feature in GitHub. The Issues feature in GitHub allows any one of our team members to track bugs, feature requests, and other tasks with detailed descriptions. There are many labels that can be assigned to issues, such as bug, documentation, duplicate, enhancement, good first issue, help wanted, invalid, question, or wontfix. These labels help the other team members know what type of issue needs to be prioritized and filter them easily. An additional feature in GitHub that allows us to organize issues and pull requests is creating milestones for the software project. Unfortunately, the team discovered this feature towards the end of the development of the project, but we started to utilize this feature as soon as the team found out about this feature. So, the team only has a milestone for Iteration 4 for now. But, all bugs prior to and even including the issues on GitHub were documented in an Excel Spreadsheet, which was also provided to the stakeholders. The milestone feature in GitHub issues helps each and every team member know what has to be done for the next goal that we are trying to reach in the project. Another practice that the team has started to follow is putting detailed descriptions in each of the issues so that the other team members can fully comprehend the problem and help out. Some issues might require a picture or another type of document in order for the team to understand the issue. Overall, the team just tries to make the issue description as detailed as possible. During one out of the four meetings we have a week as a group, we look at the bug report Excel document and the issues section of GitHub and see if there

is anything outstanding that needs to be addressed. If a team member forgot to put their concerns in either the bug report document or put it under Issues in GitHub, they just bring it up during the meeting and we address the problem during our meeting. After the issue has been resolved, we make a Pull Request to merge the updated code into the desired destination branch, which is reviewed by another team member. Once the Pull Request has been confirmed, then the issue is resolved and closed. Overall, these practices helped us develop software efficiently, keep our code quality high, communicate effectively, and finish our project successfully.