Muscle Mind

Software Configuration Management Plan

Version <2.0>

10/28/2024

Document Control

Approval

The Guidance Team and the customer shall approve this document.

Document Change Control

|  |  |
| --- | --- |
| Initial Release: | 1.0 |
| Current Release: | 2.0 |
| Indicator of Last Page in Document: | 11 |
| Date of Last Review: | 11/3/2024 |
| Date of Next Review: | 11/6/2024 |
| Target Date for Next Update: | TBD |

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: Veronica Rivas

Customer: CS 4374 Software Construction and Bhanukiran Gurijala

Software Team Members: Ruben Martinez, Alfonso Hernandez, David Duru, Logan Armendariz, Francisco Jimenez, Caleb Lopez, Jose Legarreta

Change Summary

The following table details changes made between versions of this document

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Modifier | Description |
| 1.0 | 10/7/24 | RM and FJ | Initial establishment of SCM process. |
| 1.1 | 10/14/24 | FJ | Sections 1 & 2 |
| 1.2 | 10/15/24 | FJ | Section 3 & 4 |
| 2.0 | 10/28 | RM | Specify what constitutes Minor and Major changes |

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

MuscleMind is a game designed to combine fun trivia or quiz gameplay with physical exercise prompts for users who answer questions incorrectly. This Software Configuration Management Plan (SCMP) defines the processes and tools that will be used to manage software configurations throughout the development lifecycle of MuscleMind. It provides detailed guidelines for configuration identification, change management, auditing, and version control. The intended audience includes the development team, stakeholders such as project managers and the customer, and any future team members who need to understand the configuration process.

The SCMP includes several key sections: Introduction, which provides an overview of the project, the purpose of the SCMP, and its intended audience. Please note that throughout this document the numbers of the subsections do not correspond to their supersections correctly. For example, in section 2, it is labeled as 1.1 but it should be 2.1, please disregard.

The Software Configuration Identification section explains how items like source code, design documents, and tests are organized, labeled, and backed up. The Software Configuration Control section provides an in-depth explanation of how changes to configuration items are managed. It introduces the Change Request (CR) process, which involves documenting change proposals, assessing their impact, and obtaining approvals before implementation. The role of the Configuration Control Board (CCB) is highlighted, detailing its responsibility in evaluating and approving changes to ensure they align with project goals and do not introduce conflicts or risks. This section also describes how Git is used for version control, including procedures for branching, modification, and merging to maintain a controlled and traceable development environment. The Software Configuration Auditing section focuses on ensuring compliance and system integrity. It outlines the verification process, including the use of a traceability matrix, code reviews, and automated testing, to ensure that changes meet requirements and do not introduce regressions. Additionally, it discusses the validation process, such as user acceptance testing (UAT), to confirm that the system meets customer needs and performs as expected. These sections work together to establish a robust framework for managing the configurations of the MuscleMind project, supporting its development and ensuring the delivery of a high-quality product.

## References

IEEE Standard for Software Configuration Management Plans (IEEE 828)

GitHub Repository for MuscleMind (GitHub)

https://github.com/riddle-me-ruben/muscle-mind

# Software Configuration Identification

## Software Configuration Item Identification

## The following elements will be controlled as part of the configuration:

## Source Code: Game logic, subsystems for Quiz Management, Exercise Challenges, User Management, and Performance Reporting

## Design Documents: Architectural diagrams, class diagrams, and system designs

## Test Suites: Unit tests for each subsystem and integration tests

## Requirements Documents: Software requirements specification (SRS), Design Document (SDD)

## Project Plans: Sprint plans, roadmaps, and milestones

## Project Standards: Coding standards and documentation guidelines

## User Guides: Documentation for users on how to interact with the game

## Software Configuration Item Organization

Labeling Scheme:

Each configuration item will be labeled using a versioning format. Baselines will be indicated by major version numbers (e.g., v1.0), and updates or changes to these baselines will increment by minor version numbers (e.g., v1.1 for updates). For example, v1.0 is a baseline, and v1.5 is baseline v1.0 with five updates.

### Major Changes (Increase version by 1, e.g., v1.0 to v2.0)

1. **Quiz Feature Addition**

Implementing a major quiz feature addition includes significant updates to expand the app’s core quiz functionality. Examples of this include enabling open-ended quizzes, where users enter free-form responses instead of choosing from multiple-choice options, and introducing a “Share Quiz” feature, allowing users to share quizzes they create with others. Adding open-ended quizzes requires redesigning the quiz interface to support text input, developing new evaluation logic for flexible answers, and adapting the scoring system for non-standard responses. The “Share Quiz” feature involves creating mechanisms for link-sharing, adding access controls, and ensuring quiz visibility for shared users while maintaining privacy. These types of changes make the app more interactive, flexible, and collaborative, requiring substantial updates to code, database structure, and user interface to support these new capabilities.

1. **Database Structure Change**

Changing the database structure involves altering how data is stored, related, or accessed, such as restructuring tables to support more complex relationships between quizzes, questions, and user performance tracking. For example, converting a simple table design into a relational model where each question is tied to specific quiz types or categories. This update may also impact query performance, backup protocols, and require migration scripts to ensure a smooth transition without data loss. Database changes affect core data handling and are essential for adding new features or improving efficiency in data retrieval.

### Minor Changes (Increase version by 0.1, e.g., v1.0 to v1.1)

1. **Bug Fixes**

Bug fixes include resolving errors that affect user experience or functionality but do not add new features. This can involve correcting issues like broken buttons, incorrect question sequences, missing or mislabeled scores, or logic errors in question responses. These fixes ensure that existing features work as intended, creating a smoother and more reliable user experience without major redesigns.

1. **UI Adjustments**

UI adjustments include refining the visual elements to make the app more user-friendly and aesthetically pleasing. This could involve updating color schemes, adjusting button placements, resizing fonts, or adding icons to improve clarity. These small changes enhance user experience by making the interface intuitive and visually consistent without affecting the underlying functionality.

1. **Performance Improvements**

Performance improvements focus on optimizing the app’s responsiveness, such as reducing load times, improving the speed of quiz response validation, or streamlining database queries for faster data retrieval. These changes make the app faster and more efficient, which improves the overall user experience, particularly for users with slower connections or devices, without altering features or major functionality.

This structured approach helps prioritize changes based on their impact, keeping updates consistent and manageable.

Source Code Directory Structure:

/src: Contains all source code files

/docs: Contains design documents, user guides, and configuration files

/test: Test suite files

Backup Plan:

The project database will be stored on a shared team server with daily backups. Backups will be the responsibility of Ruben Martinez (RM) and will be done using full backups every Sunday and incremental backups each night. Backup data will be stored in cloud-based storage with recovery procedures in place. Team members will work on different computers and upload their changes to the shared database, maintaining a centralized and up-to-date repository.

We will use GitHub to manage the project database and track configurations. Each team member will work on their assigned branches, which allows for parallel development without conflicts. GitHub will manage version history, so there is no need for file-based version naming like v1.0.py, which is not standard industry practice. Instead, changes will be documented through commits and pull requests, providing a clear history of modifications and updates.

The backup plan involves daily backups, with Ruben Martinez (RM) responsible for full backups every Sunday and incremental backups each night. These backups will be securely stored in cloud-based storage to ensure accessibility and data protection. In case of any data loss or corruption, recovery procedures will involve restoring the most recent backup. The team will follow a documented recovery protocol to ensure the system is backed up efficiently and with minimal disruption.

# Software Configuration Control

## Documentation

Changes to any configuration items will follow a formal process involving the use of a **Change Request (CR) form**. The CR form will include the following information:

**Requester’s Information**: Name of the individual making the request and the date of submission. This ensures accountability and provides a point of contact for further clarifications.

**Description of Change**: A clear and concise summary of the requested change, specifying which configuration items (e.g., source code, documents, or tests) will be impacted. This helps the team understand the scope and nature of the change.

**Priority Level**: Categorizes the change as High, Medium, or Low based on its potential impact on the system. High-priority changes may involve critical fixes, while low-priority changes could be enhancements or minor adjustments.

**Business Justification**: An explanation of why the change is necessary and how it aligns with the project’s goals. This provides a rationale for the change, ensuring that only relevant and beneficial modifications are considered.

**Estimated Effort**: An approximate calculation of the time required for implementation, testing, and review, expressed in hours or days. This helps in planning and allocating resources effectively.

**Initial Impact Assessment**: Completed by the development team or Configuration Control Board (CCB), this section specifies the potential risks and areas impacted by the change. It includes a risk analysis to determine if the change could disrupt other areas of the project or introduce new vulnerabilities.

**Approval Section**: Contains signatures from both the development team and the CCB to formally approve the change. Approval is only granted after thorough review and assessment, ensuring all stakeholders agree on the change's necessity and feasibility.

**Completion Details**: Documents the actual start and delivery dates for the change, along with a summary of the actual impact, comparing it with the estimated effort and risks initially assessed. This section helps in evaluating the accuracy of the estimation and the effectiveness of the implementation.

Once a CR is submitted, the change will be reviewed by the Configuration Control Board (CCB), and only approved changes will be implemented. Each change will be recorded in the Change Summary section of this document.

## Configuration Control Board

The Configuration Control Board (CCB) is responsible for the evaluation and approval or disapproval of any changes to configuration items. The CCB for MuscleMind includes all team members, and decisions regarding changes will be made collectively to ensure all stakeholders are aligned. Only authorized team members will have the right to submit, evaluate, and approve changes, maintaining control and preventing unauthorized modifications.

The CCB will meet regularly to review pending change requests. When assessing a change, the board will consider:

**Impact on system performance and stability**: How does this change affect the core functionalities of the system?

**Effect on ongoing development**: Will this change conflict with other ongoing tasks or sprints?

**Risk assessment**: Does this change introduce any potential new risks or vulnerabilities?

**User Experience**: Does this change improve or hinder the intended user experience?

**Change Approval Workflow:**

1. **Submission**: Only authorized team members submit a CR form with a detailed description of the proposed change.
2. **Initial Review**: The CCB performs an initial assessment to determine whether the change is necessary and feasible.
3. **Impact Assessment**: The CCB evaluates how the change will impact different parts of the system (e.g., the Quiz Management subsystem, Exercise Challenge system, etc.).
4. **Approval/Disapproval**: If approved, the change is assigned to the relevant team members for implementation. Disapproved changes are documented with reasons and potentially rescheduled for later review.
5. **Merge & Distribution**: Approved changes are merged into the main branch (or configuration baseline), and a notification is sent to the team. Updated artifacts will be stored in the repository.

The **CCB Approval Criteria**:

In the Muscle-Mind project, the CCB Approval Criteria distinguishes between minor and major changes to ensure clear responsibility and accountability in system modifications. Minor changes are defined as simple bug fixes, refactoring, or other non-critical updates that do not affect the core functionality of the system. These changes require approval from at least two team members before they are implemented. Major changes, on the other hand, involve modifications that impact core features or require changes across multiple subsystems. These types of changes must be approved by the entire CCB. Additionally, only authorized team members can modify specific parts of the system relevant to their responsibilities. No one may alter another team member’s component without explicit approval from the CCB or the team member responsible. Version numbers will be incremented accordingly: major changes will increase the version number by a whole number, while minor changes will increase it by 0.1.

### A major change is any modification that impacts core features or requires changes across multiple subsystems of the platform. This could include adding new functionality that changes the system’s behavior, altering the architecture or integration between subsystems, or modifying key features central to the system’s user experience. Such changes typically affect the overall operation of the system and require comprehensive testing to ensure stability. Major changes will result in the version number being incremented by 1.

### A minor change is any update that does not affect the core functionality of the system. These changes usually involve simple bug fixes, refactoring code for readability or performance improvement, or other minor adjustments that have limited impact on the system’s operation. Minor changes are considered non-disruptive and can typically be approved by two team members. Upon approval and implementation of a minor change, the version number will be incremented by 0.1.

**Verification and Validation (V&V)** will report errors through GitHub’s issue tracking system, where detailed reports including error logs, screenshots, or descriptions will be submitted. Only team members assigned to the relevant components can handle these issues to maintain consistency and accountability. Changes are distributed by the CCB after approval; they ensure that all modifications are integrated into the main branch and documented properly in the current working version directory, while previous versions are archived appropriately for reference.

## Procedures

For version control, we will use Git to manage all configuration items and track changes effectively, and GitHub so that we can all be on the same page. Team members will follow these steps:

1. **Checkout**: Each member will create a feature branch from the main branch for any new work, such as developing a feature or fixing a bug. This isolates changes and prevents conflicts in the shared codebase.
2. **Modification**: All changes should be made locally within the feature branch. Team members must document their changes through clear commit messages, detailing the modifications made and the reason behind them. This practice maintains traceability and provides context for future reviews.
3. **Approval**: Before a feature branch is merged into the main branch, it must be reviewed and approved by at least one other team member. This peer review process ensures that code quality standards are maintained, and potential issues are caught early.
4. **New Baseline**: Once changes are approved, the feature branch is merged into the main branch, which updates the configuration baseline. A new baseline version is created in the GitHub repository, documenting the version change and the specific features or fixes included.

The **Database Administrator (DBA)**, Ruben Martinez (RM), is responsible for managing access to the database and ensuring that all configurations are appropriately backed up. RM will also verify that each team member’s setup with the repository is correct and that they have the necessary permissions to create and manage their feature branches. During the sprint schedule meeting, RM will assist team members in setting up their GitHub access to the repository, ensuring everyone is aligned with the project’s version control procedures.

To create a new baseline, team members must submit a pull request (PR) from their feature branch to the main branch in GitHub. The Configuration Control Board (CCB) will review the PR. If the change is minor, approval from one team member is sufficient. For major changes, approval from the entire CCB is required. Upon approval, the PR is merged, and a new release version is documented. The DBA, RM, will manage the database and confirm the updates are consistent with the project’s backup and recovery protocols.

These procedures and tools are documented in the project’s repository for transparency and training purposes. New team members will be guided through these steps to familiarize themselves with the version control system and procedures. The GitHub repository will be maintained with version tags that clearly indicate baseline versions and significant updates, ensuring new and existing members can trace the project’s development history effectively.

# Software Configuration Auditing

Software configuration auditing ensures that the software system maintains alignment with its baseline configurations and fulfills customer requirements as outlined in the **Software Requirements Specification (SRS)**. Audits will occur at regular intervals and before each major release to ensure the integrity of the system.

The Verification Process ensures that all configuration items (SCIs), baselines, and updates accurately align with the original requirements and design specified in the Software Requirements Specification (SRS). This process verifies that any changes made to the system maintain consistency with the intended functionality and design standards. Each change will undergo a structured set of verification steps:

1. **Code Review**: A rigorous code review process will be performed for each change. The code review will involve team members who are familiar with the requirements and design, ensuring the implementation is consistent with these specifications. Reviewers will look for potential regressions, adherence to coding standards, and compliance with the project’s architecture. This step is crucial for maintaining code quality and ensuring that new code integrates seamlessly with the existing system.
2. **Testing**: Automated testing is a critical component of the verification process. All changes must pass a comprehensive suite of tests, including unit tests (to verify individual components), integration tests (to confirm the interaction between components), and system tests (to validate overall system behavior). Automated tests will be integrated into the continuous integration pipeline to ensure that any issues are caught early. Test results will be documented and reviewed to verify that the functionality is maintained across all baselines and that the software system continues to perform as intended.

The **Validation Process** focuses on confirming that the system not only meets technical specifications but also fulfills the customer’s needs and requirements. This process ensures that the software accurately reflects the customer’s expectations as outlined in the SRS.

1. **User Acceptance Testing (UAT)**: UAT will be conducted by running the system with test users who represent the end-users. This process will validate that the game performs as expected and that the features work in real-world scenarios. Feedback from UAT will be documented, and any issues or deviations from expected behavior will be logged for review and correction.
2. **Requirements Compliance**: Each configuration item will be systematically checked against the original user requirements specified in the SRS. The traceability matrix will guide this process, helping the team quickly identify which components need to be validated. Any deviations or discrepancies between the requirements and the implementation will be documented, and corrective actions will be taken before final approval. This step ensures that the system evolves in a way that remains aligned with customer needs and project goals.

By combining verification and validation processes, we ensure that the current configuration of the software system consistently mirrors the intended design and meets customer requirements. These processes will be documented and tracked using the traceability matrix and test results, allowing for transparency and accountability as the project progresses. This comprehensive approach also serves as a mechanism for training new team members, ensuring they understand the importance of maintaining alignment with the baseline and requirements documentation while implementing changes.