C++ Calculator

Version 1.0

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| 09/22/2023 | 1.0 | GitHub Initialization and plan creation | Arnav Jain |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Table of Contents

[1. Introduction 4](#_Toc11132094)

[1.1 Purpose 4](#_Toc11132095)

[1.2 Scope 4](#_Toc11132096)

[1.3 Definitions, Acronyms, and Abbreviations 4](#_Toc11132097)

[1.4 References 4](#_Toc11132098)

[1.5 Overview 5](#_Toc11132099)

[2. Project Overview 5](#_Toc11132100)

[2.1 Project Purpose, Scope, and Objectives 5](#_Toc11132101)

[2.2 Assumptions and Constraints 5](#_Toc11132102)

[2.3 Project Deliverables 5](#_Toc11132103)

[2.4 Evolution of the Software Development Plan 5](#_Toc11132104)

[3. Project Organization 5](#_Toc11132105)

[3.1 Organizational Structure 5](#_Toc11132106)

[3.2 External Interfaces 6](#_Toc11132107)

[3.3 Roles and Responsibilities 6](#_Toc11132108)

[4. Management Process 6](#_Toc11132109)

[4.1 Project Estimates 6](#_Toc11132110)

[4.2 Project Plan 6](#_Toc11132111)

[4.3 Project Monitoring and Control 7](#_Toc11132112)

[4.4 Requirements Management 7](#_Toc11132113)

[4.5 Quality Control 7](#_Toc11132114)

[4.6 Reporting and Measurement 7](#_Toc11132115)

[4.7 Risk Management 8](#_Toc11132116)

[4.8 Configuration Management 8](#_Toc11132117)

[5. Annexes 8](#_Toc11132118)

# 

# Introduction

## Purpose

The purpose of the C++ Command Line Calculator project is to provide users with a lightweight, efficient, and easy-to-use tool for performing arithmetic operations from the command line interface (CLI). This tool is designed to be versatile, powerful, and usable in most operating systems. The primary motivation behind developing this calculator is to:

1. Offer a quick method for users to perform calculations without the need for a graphical interface or external applications.
2. Provide an educational resource for students to understand the fundamentals of C++ programming and CLI application development.
3. Demonstrate good practices in software design, development, and testing using the C++ programming language.

The calculator will support basic operations such as addition, subtraction, multiplication, and division as well as more complex operations such as exponents and modulus. Furthermore, it will have error handling for cases such as missing operators, dividing by zero, etc. Further, it will be designed with extensibility in mind, allowing for potential future enhancements like support for advanced mathematical functions, unit conversions, or integration with other tools.

## Definitions, Acronyms, and Abbreviations

This section provides definitions, acronyms, and abbreviations relevant to the Arithmetic Expression Evaluator project, ensuring clear communication among stakeholders and team members.

**Definitions:**

* + **Arithmetic Expression**: A mathematical notation involving numbers and operation symbols that denotes a single mathematical operation.
  + **Operator Precedence**: The rule dictating the order in which operations in arithmetic expressions are carried out.
  + **Tokenization**: The process of converting an input string into discrete units, called tokens, for easier parsing and processing.

**Acronyms:**

* **UPEDU**: Unified Process for Education - A simplified version of the Rational Unified Process (RUP) tailored for educational projects.
* **RUP**: Rational Unified Process - A software development process framework.
* **PEMDAS**: Parentheses, Exponents, Multiplication and Division (from left to right), Addition and Subtraction (from left to right) - A mnemonic for the order of operations in arithmetic.
* **CLI**: Command-Line Interface - A user interface where users interact with software by typing commands.
* **GUI**: Graphical User Interface - A visual interface where users interact with software using graphical elements.

**Abbreviations:**

* **TA**: Teaching Assistant - A person, typically a graduate student, who assists a professor in teaching a course.
* **PM**: Project Manager - The individual responsible for overseeing the project.
* **RE**: Requirements Engineer - The individual responsible for gathering and documenting project requirements.
* **DA**: Design Architect - The individual responsible for converting requirements into a detailed software design.
* **IE**: Implementation Engineer - The individual tasked with primary coding responsibilities.
* **TE**: Test Engineer - The individual responsible for developing and executing test plans.
* **DM**: Deployment Manager - The individual overseeing the software's deployment.
* **QA**: Quality Assurance - The process of ensuring a product meets specified quality standards.

## References

1. Kruchten, P. (2003). The Rational Unified Process: An Introduction (3rd ed.). Addison-Wesley.
2. Stroustrup, B. (2013). The C++ Programming Language (4th ed.). Addison-Wesley Professional.

## Overview

This *Software Development Plan* contains the following information:

Project Overview  — provides a description of the project's purpose, scope, and objectives.  It also defines the deliverables that the project is expected to deliver.

Project Organization  — describes the organizational structure of the project team.

Management Process  — explains the estimated cost and schedule, defines the major phases and milestones for the project, and describes how the project will be monitored.

Applicable Plans and Guidelines — provide an overview of the software development process, including methods, tools and techniques to be followed.

# Project Overview

## Project Purpose, Scope, and Objectives

**Purpose:**

For our EECS 348: Software Engineering term project, we've undertaken the challenge of developing an Arithmetic Expression Evaluator in C++. Our objective isn't just to create a functional tool but to immerse ourselves in the structured and systematic approach of software development, embodying the essence of software engineering.

**Scope:**

Our Arithmetic Expression Evaluator aims to provide the following capabilities:

1. Parse arithmetic expressions input by users, ensuring operator precedence and parentheses are appropriately considered.
2. Support essential arithmetic operations (and do them according to PEMDAS order), namely:
   1. Addition (+)
   2. Subtraction (-)
   3. Multiplication (\*)
   4. Division (/)
   5. Modulo (%)
   6. Exponentiation (^)

We're also gearing up for a potential future update where we might integrate the '\*\*' operator for exponentiation, simulating a real-world change request scenario.

1. Accurately handle expressions using parentheses to determine the correct order of evaluation.
2. Detect and compute numeric constants present within the expressions.
3. Support for Unary operations
4. Offer a user-friendly command-line interface, ensuring a seamless experience for users to input expressions and retrieve results.
5. Implement comprehensive error-handling mechanisms, especially for common pitfalls like division by zero and erroneous expression inputs.

**Objectives:**

**Holistic Development Approach:** Our journey isn't just about the final software tool. We're diving deep into every phase of software development, crafting a detailed project plan, aligning our design document with stipulated requirements, and drafting rigorous test cases.

**Mastering Parsing Techniques:** This project serves as a platform to refine our understanding of parsing techniques, familiarize ourselves with data structures like stacks and trees, and enhance our algorithm design skills.

**Operational Excellence:** We aim for our evaluator to impeccably handle operator precedence (PEMDAS) and adhere to established mathematical conventions.

**Embracing Adaptability:** While our initial design caters to integer inputs, we're keeping our doors open for future enhancements, possibly including floating-point inputs.

**Ensuring Reliability:** To vouch for the reliability of our evaluator, we're accompanying it with a suite of unit tests.

**Commitment to Documentation:** We're ensuring that every piece of code we write is accompanied by clear documentation, simplifying future modifications and aiding peers in understanding our logic.

By the end of this project, we hope to present a meticulously documented C++ program that fulfills the set criteria, supplemented with essential software engineering artifacts and a comprehensive user manual.

## Assumptions and Constraints

**Assumptions:**

1. **User Input**: We assume that users have a basic understanding of arithmetic expressions. While we have error-handling mechanisms, the software relies on users entering expressions that are mathematically valid.
2. **Numeric Constants**: In the initial phase, we're assuming input to be integers. Floating-point values, while considered for future enhancements, are not part of the current scope.
3. **Operator Set**: We're assuming the set of arithmetic operators (+, -, \*, /, %, ^) to be fixed for the current version. Although we're preparing for the introduction of '\*\*' as an exponentiation operator, no other operators are being considered currently.
4. **Error Handling**: We assume that most common errors (like division by zero or unmatched parentheses) will be encountered. However, extremely edge-case scenarios might not be handled in the initial release.

**Constraints:**

1. **Development Time**: As this is a term project, we're constrained by the timeline of the semester. This limits the extent of features and optimizations we can introduce.
2. **Software Complexity**: Given the scope of the project and our academic commitments, there's a limit to the software's complexity. While we aim for a robust tool, certain advanced features or optimizations might be deferred to future iterations.
3. **Tools and Libraries**: We're working with standard C++ libraries and avoiding external dependencies to ensure portability and ease of grading.
4. **User Interface**: Our evaluator will be command-line based. A graphical user interface (GUI) is outside the current project's scope.
5. **Memory and Performance**: While our tool aims to evaluate standard arithmetic expressions efficiently, extremely long or complex expressions might impact performance due to memory and computational constraints.
6. **Future Enhancements**: While we're open to embracing changes (like the introduction of floating-point values or the '\*\*' operator), the timeline for such enhancements is uncertain. They're contingent on project progress and feedback.

By acknowledging these assumptions and constraints, we aim to set realistic expectations for our software tool and guide our development process effectively.

## Project Deliverables

Given the objectives and scope of our Arithmetic Expression Evaluator project for EECS348: Software Engineering, we anticipate the following deliverables:

1. **Project Management Plan**: A comprehensive document that outlines our project timeline, roles and responsibilities, resources, and risk management strategies. It will serve as our roadmap throughout the development process.
2. **Design Document**: A blueprint of our software that aligns seamlessly with the specified requirements. It will cover the architectural design, data structures to be used (e.g., stacks or trees), flowcharts, and pseudo-code for major functionalities.
3. **Test Plan and Cases**: Based on the requirements and design, we'll draft a set of rigorous test cases to ensure the robustness and correctness of our evaluator. This will also include a test plan detailing the testing strategy, environment setup, and criteria for success.
4. **Arithmetic Expression Evaluator (Software)**:
   * A C++ program capable of parsing and evaluating arithmetic expressions.
   * Features as per the project scope: operator support, parenthesis handling, error management, and a user-friendly command-line interface.
   * Initial support for integer-based calculations, with a design that's adaptable for future enhancements.
5. **Code Documentation**: Inline comments within our C++ code explaining the logic, purpose, and intricacies of each function or module.
6. **User Manual/README**: A comprehensive guide on how to use our arithmetic evaluator. It will cover installation instructions, examples of valid and invalid expressions, and troubleshooting tips.
7. **Post-Development Review Document**: After the project's completion, we'll compile a review document that highlights our challenges, learnings, and recommendations for future iterations or similar projects.

By delivering on each of these components, we aim to provide a holistic view of our software development journey, demonstrating both our technical prowess and our adherence to structured software engineering principles.

## Evolution of the Software Development Plan

The Software Development Plan (SDP) is a living document. As the project progresses and new information becomes available, the SDP may need to be revised to reflect changes in scope, resources, or other unforeseen challenges. Here’s how we envision the evolution of our SDP for the Arithmetic Expression Evaluator:

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Description** | **Criteria for Unscheduled Revision** | **Reissue Date** |
| 1.0 | Initial Draft | Major discrepancies in initial requirements or fundamental misunderstandings. | September 24, 2023 |
| 1.1 | Revised after Requirement Analysis | Significant changes in stakeholder feedback or user requirements. | October 10, 2023 |
| 1.2 | Update after Design Phase | Major design changes or shifts in technology stack. | October 26, 2023 |
| 1.3 | Mid-Project Review | Significant deviations from planned progress or newly identified risks. | November 12, 2023 |
| 1.4 | Post-Testing Adjustments | Critical issues identified during testing that require major adjustments. | November 28, 2023 |
| 2.0 | Final Review before Deployment | Last-minute changes from stakeholders or final testing results that mandate adjustments. | December 15, 2023 |

# Project Organization

## Organizational Structure

The organizational structure for the Arithmetic Expression Evaluator project has been carefully designed to ensure efficient communication, clear role delineation, and effective decision-making. Here's a breakdown of our project's organizational hierarchy:

**Project Team Lead (PTL):**

* + **Responsibilities**: Overarching project management, stakeholder communication, and final decision-making.
  + **Assigned to**: Arnav Jain

**Development Team:**

1. **Requirements Engineer (RE)**:
   * **Responsibilities**: Gathering, analyzing, and documenting project requirements.
   * **Assigned to**: Nabeel Ahmad
2. **Design Architect (DA)**:
   * **Responsibilities**: Creating a detailed software design and ensuring it aligns with project requirements.
   * **Assigned to**: Omar Mohammed
3. **Implementation Engineers (IEs)**:
   * **Responsibilities**: Coding, debugging, and initial unit testing.
   * **Assigned to**: Yaeesh Mukadam, Zonaid Prithu

**Support Team:**

1. **Quality Assurance (QA) & Documentation Specialist**:
   * **Responsibilities**: Conducting quality checks, ensuring code and design standards, and handling documentation.
   * **Assigned to:** Humza Qureshi, Shero Baig

**Advisory and Review:**

1. **Course Instructor & Teaching Assistants**:
   * **Responsibilities**: Offering guidance, reviewing project milestones, and providing feedback.

* **Organizational Flow:**
* The **PTL** oversees the entire project, ensuring that all teams are aligned and milestones are met.
* The **Development Team** handles the core phases of the software lifecycle: requirements, design, and coding.
* The **Support Team** ensures the software's smooth transition from development to the end-users and maintains its quality.
* The **Advisory and Review** group, consisting of the course instructor and TAs, offers external guidance, feedback, and assessment.

This organizational structure facilitates clear communication channels, ensures every team member has a defined role, and allows for efficient project progression.

## External Interfaces

1. **Course Instructors & Teaching Assistants (TAs)**:
   * **Internal Contact**: Project Team Lead
   * **External Contact**: Professor Hossein Saiedian & Assigned TA
   * **Responsibilities**: Reviewing project progress, providing feedback, and final project evaluation. Assisting with deployment guidelines and validating the acceptance criteria for the product.
2. **User Groups for Testing**:
   * **Internal Contact**: Testing Lead
   * **External Contact**: Selected student groups from EECS348
   * **Responsibilities**: Providing real-time feedback during user testing sessions. The internal team will handle deployment for testing and gather feedback for refinements.

## Roles and Responsibilities

|  |  |
| --- | --- |
| **Person** | **Unified Process for EDUcation Role** |
| Arnav Jain | Project Manager |
| Nabeel Ahmad | Requirements Engineer |
| Omar Mohammad | Design Architect |
| Yaeesh Mukadam, Zonaid Prithu | Implementation Engineer |
| Humza Qureshi, Shero Baig | Quality Assurance |

Anyone on the project can perform Any Role activities.

# Management Process

## Project Estimates

**Estimated Cost:**

Given that this is an academic project, and we'll be utilizing open-source tools and libraries, the estimated cost for this project is $0.

**Estimated Schedule:**

* **Requirement Analysis**: 2 weeks
* **Design Phase**: 2 weeks
* **Implementation Phase**: 3 weeks
* **Testing Phase**: 2 weeks
* **Deployment & Documentation**: 1 weeks

**Total Duration**: 10 weeks

**Basis for Estimates:**

1. **Course Guidelines**: The time frame provided by the course and the project description.
2. **Team Availability**: Considering the team's academic commitments and the end-of-semester timeline.

**Re-estimation Points:**

1. **Post-Requirement Analysis**: To adjust for any scope changes or clarifications.
2. **Mid-Implementation**: Evaluating the development progress and determining if any adjustments to the timeline are necessary.
3. **After Initial Testing**: To account for potential bug-fixing or refinements before the final submission on December 6th.

## Project Plan

### Phase Plan

The C++ Calculator project is strategically divided into multiple phases to ensure systematic progression. Here's an overview of the project's timeline, major milestones, and significant release points.

**Major Milestones with Achievement Criteria:**

1. **Requirement Document Completion**:
   * **Criteria**: All project requirements, including functional and non-functional, are detailed and documented. Feedback from the team and initial stakeholders has been incorporated.
2. **Design Document Completion**:
   * **Criteria**: The software's architectural design, data structures, and main algorithms are outlined. The team agrees upon the design principles and methodologies to be used.
3. **Alpha Version Release**:
   * **Criteria**: The first working version of the software is released. It encompasses the core functionalities and is ready for internal testing.
4. **Beta Version Release**:
   * **Criteria**: Post alpha testing, the refined version of the software is released. It is more stable and includes fixes for the issues identified during alpha testing.
5. **Final Product Deployment and Documentation Release**:
   * **Criteria**: The final and stable version of the software is deployed, and all documentation (including user manuals and technical documents) is made available.

### Iteration Objectives

Iterations in UPEDU are time-boxed cycles during which specific objectives are targeted. These objectives align with the broader project goals and are used to progressively build the software, ensuring regular feedback and adaptability. Here are the objectives for each iteration of the Arithmetic Expression Evaluator project:

* **Iteration 1: Requirement Analysis**

**Objective**: Establish a clear understanding of the project's scope and requirements.

* + Gather detailed functional and non-functional requirements.
  + Prioritize requirements based on importance and dependencies.
  + Document the requirements and get initial stakeholder feedback.
* **Iteration 2: Design**

**Objective**: Create the foundational design for the software.

* Outline the software's architecture.
* Design the primary data structures and algorithms.
* Document the design decisions and get team consensus.
* **Iteration 3: Initial Implementation (Alpha Development)**

**Objective**: Develop the core functionalities of the software.

* Implement expression parsing and tokenization.
* Add support for arithmetic operators (+, -, \*, /, ^, %) as well as parenthesis.
* Develop the basic command-line interface.
* **Iteration 4: Refinement (Beta Development)**

**Objective**: Refine the software based on feedback and testing outcomes.

* Address bugs and issues identified during alpha testing.
* Optimize performance and enhance the user interface.
* Implement robust error handling.
* **Iteration 5: Deployment & Documentation**

**Objective**: Release the final product and ensure users have the necessary resources to use it effectively.

* Deploy the software for end-users.
* Release comprehensive documentation, including a user manual and technical documentation.
* Gather feedback post-deployment for potential future iterations.

A graph with blue rectangles

Description automatically generated

Each iteration aims to add value to the project, build on the previous iteration's work, and ensure that the software is steadily progressing towards the final product. Regular reviews at the end of each iteration ensure alignment with objectives and provide opportunities for course correction if necessary.

Note: This schedule is tentative and subject to change based on due dates

### Releases

1. **Alpha Release**:
   * **Type**: Demo
   * **Description**: Initial working version of the software, showcasing core functionalities and design. Suitable for internal testing and feedback.
2. **Beta Release**:
   * **Type**: Beta
   * **Description**: Refined version post alpha feedback, more stable with additional features. Ready for a wider audience testing, including potential end-users.
3. **Final Product Release**:
   * **Type**: Production
   * **Description**: Stable and polished version of the software, incorporating feedback from all testing phases. Ready for deployment and general use.

### Project Schedule

|  |  |  |
| --- | --- | --- |
| Activity/Phase/Milestone | Start Date | End Date |
| Requirement Analysis | Sep 10, 2023 | Sep 24, 2023 |
| Design Phase | Sep 25, 2023 | Oct 08, 2023 |
| Implementation (Alpha) | Oct 09, 2023 | Oct 29, 2023 |
| Testing (Alpha) | Oct 30, 2023 | Nov 12, 2023 |
| Refinement (Beta) | Nov 13, 2023 | Dec 09, 2023 |
| Deployment & Documentation | Dec 10, 2023 | Dec 15, 2023 |
| Releases |  | **Date** |
| Alpha Release (Demo) |  | Oct 29, 2023 |
| Beta Release |  | Nov 13, 2023 |
| Final Product Release |  | Dec 15, 2023 |

Note: This schedule is tentative and subject to change based on due dates

### Project Resourcing

**Requirement Analysis:**

* **Staff Required**: 2
  + **Roles**: Project Team Lead, Requirements Engineer
  + **Skills**: Stakeholder communication, requirements elicitation, and documentation

**Design:**

* **Staff Required**: 2
  + **Roles**: Design Architect, Requirements Engineer (for design validation)
  + **Skills**: System design, data structure and algorithm design, UML modeling

**Initial Implementation (Alpha Development):**

* **Staff Required**: 3
  + **Roles**: Implementation Engineers (2), Design Architect (for code validation)
  + **Skills**: C++ programming, debugging, basic CLI design

**Testing & Feedback (Alpha Testing):**

* **Staff Required**: 4
  + **Roles**: Quality Assurance (2), Implementation Engineer (2, for addressing immediate bugs)
  + **Skills**: Test case design, unit testing, integration testing

**Refinement (Beta Development):**

* **Staff Required**: 3
  + **Roles**: Implementation Engineers (2), Test Engineer (for validation)
  + **Skills**: C++ programming, debugging, optimization

**Final Testing & Deployment Preparation:**

* **Staff Required**: 3
  + **Roles**: Quality Assurance (2), Project Manager
  + **Skills**: User acceptance testing, deployment planning, documentation

**Deployment & Documentation:**

* **Staff Required**: 1
  + **Roles**: Project Manager
  + **Skills**: Software deployment, user manual creation, technical documentation

Throughout the project, the **Project Team Lead** will be overseeing the entire process, ensuring milestones are met and coordinating with the team and external stakeholders.

Note: Given the nature of academic projects, flexibility is essential. Team members might occasionally wear multiple hats or collaborate closely on tasks, ensuring that the project progresses smoothly.

## Project Monitoring and Control

To ensure the C++ Calculator project progresses efficiently and meets the specified requirements, we've established the following monitoring and control mechanisms:

**Requirements Management:**

* **Information Collection**: Regular meetings will be held to discuss the progression against requirements. Any changes or deviations will be documented.
* **Control Mechanisms**: A version-controlled requirement document will be maintained. Any changes to the requirements will go through a review process.

**Quality Control:**

* **Timing & Methods**: Quality checks will be done at the end of each phase. This includes code reviews, design evaluations, and requirement validation.
* **Evaluation Techniques**:
  + **Walkthroughs**: Team members will present their work to peers for feedback.
  + **Inspections**: Detailed examinations of project artifacts, such as code or design documents, will be conducted.
  + **Reviews**: Regular reviews with stakeholders, including the course instructor and TAs, will be held to ensure the project aligns with expectations.

**Reporting and Measurement:**

* **Reports**:
* **Weekly Progress Report**: Details the work done during the week, any challenges faced, and plans for the upcoming week.
* **Milestone Completion Report**: Generated upon the completion of each major project milestone, detailing achievements and any deviations from the plan.
* **Metrics**:
  + **Bug Tracking**: Number of bugs reported and resolved.
  + **Code Churn**: Measures how much code is being changed over time.
  + **Completion Percentage**: Tracks the percentage of tasks completed versus planned tasks.
* **Risk Management:**
* **Approach**:
  + **Identification**: Potential risks, both technical and non-technical, will be identified at the beginning of the project.
  + **Analysis**: Each risk will be analyzed for its potential impact and likelihood of occurrence.
  + **Prioritization**: Risks will be ranked based on their potential impact and likelihood.
  + **Monitoring**: Regular check-ins will be conducted to monitor identified risks and any new risks that arise.
  + **Mitigation**: For each risk, a mitigation strategy will be developed and implemented if the risk materializes.
* **Configuration Management:**
* **Problem & Change Submission**: A shared platform, like a version control system or project management tool, will be used to track issues and changes.
* **Artifact Naming & Numbering**: A standard naming convention will be established for all project artifacts, ensuring consistency and clarity.
* **Retention & Backup**: All project artifacts will be stored in a version-controlled repository, ensuring that changes can be tracked and reverted if necessary. Regular backups will be taken, and a disaster recovery plan will be in place.

For more detailed information on specific metrics, risks, and configuration management procedures, please refer to the respective detailed documents, such as the Project Measurements and Configuration Management Plan documents.

 [The following is a checklist of items to consider:

* Requirements Management: Specify the information and control mechanisms which will be collected and used for measuring, reporting, and controlling changes to the product requirements.
* Quality Control: Describe the timing and methods to be used to control the quality of the project deliverables and how to take corrective action when required. Include techniques, metrics, criteria, and procedures used for evaluation— this will include walkthroughs, inspections, and reviews. Note that this is in addition to the Test Plan, which is not enclosed in the Software Development Plan.
* Reporting and Measurement: Describe reports to be generated. Specify which metrics should be collected and why. **OR** if available, refer to the **Project Measurements and Project Measurements** document
* Risk Management: Describe the approach that will be used to identify, analyze, prioritize, monitor and mitigate risks. If available, refer to the **Risk List** document.
* Configuration Management: Describe the process by which problems and changes are submitted, reviewed, and dispositioned. Describe how project or product artifacts are to be named, marked, and numbered, including system software, plans, models, components, test software, results and data, executables, and so on. Describe retention policies, and the back-up, disaster, and recovery plans. **OR** if Available, Refer to the **Configuration Management Plan** document

The text that follows is provided as an example.]

## ****Requirements Management****

**We will hold regular meetings in order to discuss and collect information regarding the requirements and will document any changes in a version-controlled document. Any changes in requirements will go through a rigorous review process.**

## ****Quality Control****

**The quality checks will be done at the end of each phase. We will have walkthroughs, inspections, and reviews in order to ensure quality. At walkthroughs, team members will present their work to the rest of the team and gather feedback. At inspections, detailed examinations of the artifacts will occur. Finally, reviews will be held with the stakeholders (teacher and TAs) to ensure project quality.**

## ****Reporting and Measurement****

**We will hold weekly progress reports between the team where we will detail all the work done for the given week and outline the plan for the next week. Outside of this, we will also hold major progress reports at the end of each milestone completion where we will discuss the milestone, the achievements, and any deviations from the plan.**

**The metrics used for the project are bug tracking, code churn, and completion percentage. Bug tracking accounts for the number of bugs reported and resolved. Code churn measures how much of the code is being changed. Finally, the completion percentage measures how far along the project we are at.**

## ****Risk Management****

**We will begin with identifying potential risks at the beginning of the project. Each potential risk will then be analyzed for its likelihood of occurring and cost of occurrence. We will then create a ranking of the risks based on the analysis and prioritize risks that have a higher chance of happening with higher potential harm. We will perform regular check-ins at meetings to monitor the risks and analyze if any new potential risks have arisen. For each risk, we will implement a mitigation strategy to eliminate or minimize the risk.**

## ****Configuration Management****

We will utilize a shared platform that will be used to collaborate by the team and to track issues and

changes. A standard naming convention will be implemented for all artifacts in order to ensure consistency,

simplicity, and ease for the entirety of the project. Also, all the project artifacts will be stored in a remote version control system (Github) that all the team members have access to. This will ensure all changes are tracked and stored in case a reversion is necessary. Regular backups will be kept, and a disaster recovery plan will be in place.

# Annexes

The project will follow the UPEDU process.

Other applicable process plans are listed in the references section, including Programming Guidelines.