Team 14

Arithmetic Expression Evaluator in C++
Software Development Plan
Version 1.0

Revision History

Date	Version	Description	Author
29/09/2024	1.0	Original Draft	Team 14

Table of Contents

Software Development Plan

1. Introduction

This Software Development Plan outlines the framework for an Arithmetic Expression Evaluator in C++. Our primary object with this project is to design and implement a robust arithmetic expression parser that will serve as a component of a larger compiler product for language L that is being developed in C++. The parser that we design will be able to parse and evaluate expressions that include arithmetic operators such as addition, subtraction, multiplication, division, modulo, and exponentiation, as well as manage operator precedence and grouping parentheses.

The scope of this project encompasses the software development lifecycle, which includes a comprehensive project plan, a requirements document, and a design document that aligns with the requirements. Each stage of the project (requirements, design, implementation, testing) will be interlinked to ensure coherence and to adhere to the outlined objectives. The development process will also involve rigorous test cases to validate the functionality of the evaluator against the requirements.

In order to facilitate a clear understanding of this project, this document also provides definitions of key terms, acronyms and abbreviations, as well as references in the following sections.

1.1 Purpose

The purpose of the *Software Development Plan* is to gather all information necessary to accurately guide the development of an arithmetic expression evaluator in C++. It describes the approach to the development of this expression evaluator and is the top-level plan discussed and agreed upon by all members of the development team.

The following people use the *Software Development Plan*:

- The **Project Managers** use it to plan the project schedule and resource needs, and to track progress against the schedule.
- The **Technical Lead** uses it to help guide what technical advice they offer the team, along with identifying what coding practices are most important to keep in mind throughout development.
- The **Quality Assurance Leads** use this document to identify what standards all other project artifacts must meet. This document will also be used by QA leads to identify specific, important test cases to account for.
- The **Product Owner** uses this document to ensure all user stories and requirements are accounted for in the project plan.
- The **Scrum Master** uses this document to organize sprints in accordance with the slated timeline and schedule agreed upon by all members of the development team.

1.2 Scope

This *Software Development Plan* describes the overall plan to be used by the Arithmetic Expression Evaluator in C++ project, including deployment of the product. The details of the individual iterations will be described in the Iteration Plans.

The plans as outlined in this document are based upon the product requirements as defined in the *Requirements Document.*

The Arithmetic Expression Evaluator in C++ project will be integrated into a larger compiler. The details and planning of this larger compiler is not within the scope of this *Software Development Plan.*

1.3 Definitions, Acronyms, and Abbreviations

- **PEMDAS**: Parentheses, Exponents, Multiplication, Division, Addition, Subtraction - This acronym refers to the order of operations that must be followed when evaluating arithmetic expressions.
- **Parser**: A program or component that interprets and processes input data according to rules.
- **Expression Tree**: A binary tree used to represent the structure of an arithmetic expression where internal nodes are operators and leaf nodes are operands.
- **Object-Oriented Programming**: A programming paradigm based on the concept of objects, which contain data and methods. C++ is the language used in this project, and it supports OOP principles like encapsulation, inheritance, and polymorphism.
- **Command-Line Interface**: A text-based user interface where users input commands to interact with the software. Our project will utilize a CLI for inputting expressions and displaying results.

1.4 References

- **Iteration Plans**: Document that outlines the planned iterations, goals, and deliverables for each phase of the project. Available through the project's internal documentation system.
- **Vision**: Create a reliable and efficient parser that accurately processes and evaluates arithmetic expressions in C++.
- **Glossary**: A glossary of terms related to the project, including key definitions of technical terms, abbreviations, and acronyms. Accessible in the project repository.
- **Software Development Plan Template**: This document follows a standard Software Development Plan template and is used to guide a team.

1.5 Overview

This *Software Development Plan* contains the following information:

Project Overview — Provides a description of the Arithmetic Expression Parser project's purpose, scope, and objectives. It defines the functionality the parser is expected to deliver as part of the larger compiler for C++.

Project Organization — The project team is organized into key roles to ensure effective collaboration and accountability. It includes a Scrum Master, Project Managers (1 and 2), a Technical Lead, a Product Owner, and two QA Leads, each responsible for specific aspects of project execution and quality assurance.

Management Process — This section provides an overview of the project timeline, outlining key phases, milestones, and deliverables for the parser development. It also describes the methods for tracking and assessing project progress.

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

2. Project Overview

2.1 Project Purpose, Scope, and Objectives

Our team will create an arithmetic expression parser component of a larger compiler language in C++. The parser will be capable of parsing through and evaluating arithmetic expressions using the precedence rules of PEMDAS. Also, the program will evaluate expressions in parentheses. Expressions will include the operators: +, -, *, /, **, and % as well as numeric constants.Moreover, the parser should also be able to handle any potential errors effectively and be accessible to the user via an easily readable command line interface.

By the end of the project, our parser will deliver the following:

1. The most common software engineering artifacts, e.g.,
    - A project management plan,
    - A requirements document,
    - A design document, and
    - A test plan (test cases, expected results, actual results)

2. A well-documented C++ program that can evaluate arithmetic expressions with the specified operators and features.

3. A user manual or README file explaining how to use your program, including examples.

2.2 Assumptions and Constraints

ASSUMPTIONS:

1: All team members have access to their own personal computers with a C++ compiler and are familiar with the language.

2: There will be enough time allocated to design, implement, and test the project.

3: Documentation and artifacts will be provided for every step of the development cycle

CONSTRAINTS:

1: There are 7 available team members to work on this project.

2: The project must be submitted by the deadline

2.3 Project Deliverables

PROJECT PLAN: 9/29

REQUIREMENTS: 10/13

ARCHITECTURE/DESIGN: 11/10

TEST CASES: 12/12

USER MANUAL: 12/12

PROJECT CODE: 12/12

Deliverables for each project phase are identified in the Development Case. Deliverables are delivered towards the end of the iteration, as specified in section *4.2.4 Project Schedule*.

2.4 Evolution of the Software Development Plan

Version	Date	Revision	Criteria for revision
1.0	9/29/2024	Initial Draft	N/A
2.0	10/02/2024	Requirements Engineering	Add new requirements and requests
3.0	10/09/2024	Software Design	Add development timeline with sprints
4.0	11/13/2024	Testing	Add testing methods and results
5.0	12/12/2024	Finalization	Refine

The *Software Development Plan* will be revised prior to the start of each Iteration phase.

3. Project Organization

3.1 Organizational Structure

The project team follows a collaborative structure designed to ensure effective management, Quality assurance, and adherence to Agile practices.

1. Project Managers (Kristoffer Comahig and Drew Franke) - Responsible for overseeing the project schedule, assigning tasks, and ensuring deadlines are met. The project managers also address any issues that arise throughout the development process.

2. Technical Lead (Axel Bengoa) - Provides technical guidance and advice to the team. This role is crucial for ensuring that best coding practices are followed and that technical challenges are addressed effectively throughout the development process.

3. Quality Assurance Leads (Owen Berkholtz and Gael Salazar-Morales) - Ensures that all project artifacts meet quality standards. This role involves planning and executing tests during the coding phase to identify and rectify bugs, thereby ensuring the functionality and reliability of the arithmetic expression evaluator.

4. Product Owner (Quinn Westrope) - Responsible for gathering and clarifying project requirements. The Product Owner ensures that all stakeholder needs are addressed and that the project aligns with the intended goals and specifications.

5. Scrum Master (Bryson Toubassi) - Facilitates the team’s adherence to Agile practices. The Scrum Master organizes daily check-ins, coordinates work sessions, and helps remove any impediments that may hinder the team's progress.

3.2 External Interfaces

3.3 Roles and Responsibilities

Person	Unified Process for EDUcation Role
Kristoffer Comahig	Project Manager - Keeps track of the project schedule, assigns tasks, and makes sure everyone meets deadlines. They also handle any issues that come up.
Drew Franke	Project Manager 2
Axel Bengoa	Technical Lead - Offers technical advice and ensures the team follows good coding practices. They also help solve any technical problems.
Owen Berkholtz	QA Lead - Makes sure the project artifacts meet quality standards. During coding, they plan and run tests to find and fix bugs.
Gael Salazar-Morales	QA Lead 2
Quinn Westrope	Product Owner - Makes sure all the requirements are in place and accounted for
Bryson Toubassi	Scrum Master - Helps the team follow Agile practices, organizes daily check-ins, and plans work sessions.

Team Member Profiles

1. Kristoffer Comahig - Project Manager
    a. Contact: kris.comahig@ku.edu
    b. Availability: when2meet
    c. Computing Platform Experience: Windows and Linux
    d. Programming Language Experience: Python (Intermediate), C++ (Beginner/Intermediate), C(Intermediate)
2. Drew Franke - Project Manager
    a. Contact: andrewfranke@ku.edu
    b. Availability: when2meet
    c. Computing Platform Experience: Windows
    d. Programming Language Experience: Python (Intermediate), C++, HTML5, CSS3 (Beginner)
3. Axel Bengoa - Technical Lead
    a. Contact: axelbengoa@ku.edu
    b. Availability: when2meet
    c. Computing Platform Experience: macOS
    d. Programming Language Experience: Python (Intermediate), Java (Beginner), HTML(Beginner)
4. Owen Berkholtz - Quality Assurance Lead
    a. Contact: owen.berkholtz@ku.edu
    b. Availability: when2meet
    c. Computing Platform Experience: Windows and Linux
    d. Programming Language Experience: Python (Intermediate), C++ (Beginner)
5. Gael Salazar-Morales - Quality Assurance Lead
    a. Contact: gmorales034@ku.edu
    b. Availability: when2meet
    c. Computing Platform Experience: Windows and Linux
    d. Programming Language Experience: Python (Intermediate), C++ (Beginner)

6. Quinn Westrope - Product Owner
    a. Contact: quinnwestrope@ku.edu
    b. Availability: when2meet
    c. Computing Platform Experience: macOS
    d. Programming Language Experience: Python, JavaScript, HTML, CSS
7. Bryson Toubassi - Scrum Master
    a. Contact: btoubassi@ku.edu
    b. Availability: when2meet
    c. Computing Platform Experience: Windows and macOS
    d. Programming Language Experience: Python (Intermediate), C, JavaScript, HTML, (Beginner-Intermediate)

4. Management Process

4.1 Project Estimates

4.2 Project Plan

The project is structured into several interactions focused on progressively building and refining the software, beginning with basic expression parsing and operator precedence handling, and culminating in a final implementation with comprehensive testing and bug fixing. The releases transition from a demo version to a beta release and finally a fully functional release. The project schedule outlines key phases and their corresponding milestones. Each one week sprint is meticulously planned to ensure timely completion. Weekly meetings are to occur every Wednesday between September 18, 2024 and December 12, 2024 at available rooms in the LEEP2 building. Communication is to take place in our Discord channel with channels for every topic relating to this project.

4.2.1 Phase Plan

4.2.2 Iteration Objectives

Iterations

1: Complete basic expression parsing and operator handling.

2: Implement parentheses evaluation and error handling.

3: Finalize user interface and add test cases.

4: Refine code, resolve any remaining bugs, and prepare for final testing.

4.2.3 Releases

Releases

1: Initial demo version focusing on basic functionality like expression parsing and operator handling.

2: Beta release which includes parentheses handling, error management, and initial testing.

3: Final release with a complete, full functionality, user interface, and overall testing.

4.2.4 Project Schedule

Phase	Start Date	End Date	Duration	Deliverables
Role Assignments and Project (Part 1)	Sep 18, 2024	Sep 29, 2024	1 week	-Roles assigned -Part 1 complete
Requirements Engineering (Part 2)	Oct 2, 2024	Oct 13, 2024	2 weeks	Requirements Doc Complete
Architecture and Design (Part 3)	Oct 16, 2024	Nov, 10 2024	3 weeks	Architecture and Design
Implementation (Part 4) Test Cases (Part 5) User Manual (Part 6)	Nov 10, 2024	Dec 12, 2024	4 weeks	-Updated project -Management Plan -Requirements -Design -Test cases -C++ code -User manual

4.2.5 Project Resourcing

4.3 Project Monitoring and Control

A Requirements Document will be created to specify information and control mechanisms. This document will detail, structure, and document the characteristics of the software as we iterate.
A weekly scrum will be held so we can measure and report different aspects of the project. These meetings will highlight the processes that are going well and those that may need to be changed. The meetings will also work to validate the functions of our program against the requirements. Verification will also be inspected to ensure that all components work together with completeness, consistency, and feasibility of the requirements.

We are communicating on a Discord server where we are also linking to project artifacts.

4.4 Requirements Management

4.5 Quality Control

Defects will be recorded and tracked as Change Requests, and defect metrics will be gathered (see Reporting and Measurement below).

All deliverables are required to go through the appropriate review process, as described in the Development Case. The review is required to ensure that each deliverable is of acceptable quality, using guidelines and checklists.

Any defects found during review which are not corrected prior to releasing for integration must be captured as Change Requests so that they are not forgotten.

The main objective in validating our requirements is Quality Assurance. As stated above, verification of the functions and validation of necessary requirements will be crucial in determining our control over quality. Walkthroughs will be held during meetings to thoroughly cover requirements and design phases. Inspections will be performed during the construction phase, where we will meet and systematically ensure that all code adheres to requirements and is free of defects.

4.6 Reporting and Measurement

4.7 Risk Management

Project risk is evaluated at least once per iteration and documented in the appropriate discord channel. Risks will be analyzed qualitatively taking factors such as project scope into consideration. Project risk will also be evaluated quantitatively using measures such as time cost. Prioritization of risks will be based on scoring of each individual risk with their likelihood and impact being given some thought. Each meeting we will monitor the status of each identified risk, identify any new risks, and assign members most closely associated with each risk to be responsible for monitoring and managing their specific risk. Mitigation plans will be required for high-priority risks and contingency plans for risks that cannot be mitigated.

4.8 Configuration Management

Github will be used to provide a database of Change Requests and a controlled versioned repository of project artifacts.

All source code, test scripts, and data files are included in baselines. Documentation related to the source code is also included in the baseline, such as design documentation. All customer deliverable artifacts are included in the final baseline of the iteration, including executables.

The Change Requests are reviewed and approved by all members of the project. All change requests must be submitted via Discord direct messages. Problem reports should also be reported this way. Members should conduct an impact analysis to understand the implications of each proposed change. Use predefined naming conventions for all project artifacts. Archive old versions of project artifacts for future reference. A backup of all critical data and software should be made in case of disaster.

5. Annexes

The project will follow the **UPEDU process**.

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

Programming Guidelines- code readability, error handling, version control, and breaking code into functions and classes