<Kansas Instruments>

<Arithmetic Expression Evaluator>
Software Development Plan

Version <1.1>

Revision History

Date	Version	Description	Author
09/17/2024	1.0	Starting the project plan, created a shared GitHub repository	Emma, Vrishank, Phoenix, Kit, Khang
09/20/2024	1.01	Added Section 4.2	Phoenix
9/21/2024	1.02	Added to section 4.1	Khang
9/22	1.03	Added to Section 3	Emma
9/23	1.04	Added to sections 2.1-2.5	Kit
9/23	1.05	Added to section 5	Jakob
9/24	1.06	Added section 1	Vrishank
9/26	1.1	Finishing Touches	Emma, Vrishank, Phoenix, Kit, Khang, Jakob, Aiden

Table of Contents

Software Development Plan

1. Introduction

This *Software Development Plan* will outline the projected plan for making an arithmetic expression parser in C++. This document includes information about the scope, project schedule, meeting times, roles, release plan of the project, and management plans for the project. The project schedule is divided into the different sections of the project. The first of which is this project plan. The project schedule includes the dates of when the finished products are projected to be pushed to GitHub. It will explain PEMDAS, GCC, and the Unified Process for EDUcation.

1.1 Purpose

The purpose of the *Software Development Plan* is to gather all information necessary to manage, guide, and build the project in the future. It will be the outline for the project management, construction, and timeline. It will do that by outlining the schedule for the release of project components, and the management process that will be used to achieve our goals.

The following people use the *Software Development Plan:*

- The **Project Manager** uses it to schedule when to meet and what to work on.
- The **Quality Tester** will use it to determine what versions of the project need to be ready to be turned in and which version can still be worked on according to the given schedule.
- The **Notetaker** will use it to see when each task should be done, and who is working on what tasks, so they can mark it in their document.
- The **Risk Management** will use it to check if anything on the schedule is unfeasible and needs to be amended, to avoid unrealistic deadlines and cause delays in the schedule.
- The **Configuration Management** will use it to make sure that each version of the project has the correct features, and is ready to be pushed.
- The **Requirements Manager** will use it to make sure that everything is updated and traceable so the team knows exactly what version they are working on and what they need to do to update the old versions.
-

1.2 Scope

This *Software Development Plan* describes the Arithmetic Expression Parser, the project organization, the schedule, the management process/roles, and the constraints for the project. It will serve as the outline for all the project components to come in the future. The details of each version are listed in the 2.5, the *Evolution of the Software Development Plan.* The schedule for the successive project components is listed in section 2.4, Project Deliverables. The Notetaker notes will be a direct result of the meeting times outlined on the schedule listed in 2.4. The plans outlined in this document are based upon the product requirements as defined in the *Vision Document*.

1.3 Definitions, Acronyms, and Abbreviations

PEMDAS: This acronym stands for Parentheses, Exponent, Multiplication/Division, Addition/Subtraction. It lists the order of operations from which the arithmetic program will rely on. This acronym is necessary to ensure that the arithmetic operations done in the program are accurate.

GCC: This stands for GNU compiler collection. This is the compile command in Unix systems. As we will be using a variety of systems, and will be using Git for version control GCC will be used throughout the project to compile the programs that we will make in the future.

UPEDU: This is an acronym for the Unified Process Education. It is a simplified version of the Unified Process, the most used process model in software engineering, designed for students.

See the Project Glossary.

1.4 References

- Vision:

  "To develop a robust and intuitive arithmetic expression evaluator that seamlessly processes complex calculations with precision and clarity, empowering users to confidently solve mathematical expressions with a user-friendly interface and reliable error handling."

Title	Report Number	Publishing Organization	Date
Vision Statement	001.1.4	Kansas Instruments	9/26/24
Iteration Plan	001.4.1.3	Kansas Instruments	TBD
Glossary	001.1.3	Kansas Instruments	TBD
Notaker Document	004	Kansas Instruments	9/17/24

1.5 Overview

This *Software Development Plan* contains the following information:

Project Overview	—	Provides a description of the project's purpose, scope, and objectives. Assumes that all members of the team can code in C++ to complete the project. Provides a schedule for the goals and deliverables for the Arithmetic Expression Parsing Project.
Project Organization	—	describes the organizational structure of the project team, who occupies what role
Management Process	—	explains the estimated schedule, defines the major phases and milestones for the project, and describes how the project will be monitored. Describes each iteration of the project, and the objectives of each iteration.
Annexes	—	provide an overview of the software development process, including methods, tools, and standards to follow

2. Project Overview

2.1 Project Purpose, Scope, and Objective

This project aims to create an arithmetic expression evaluator that can accurately parse and evaluate arithmetic expressions using the order of operation (PEMDAS). The objective is to ensure user-friendly interaction through our interface, correctly calculate the expressions, and have a way of handling errors/invalid expressions.

2.2 Assumptions and Constraints

2.3

Team members stay on schedule and are capable of coding in C++.

2.4 Project Deliverables / Iteration Plan

Deliverables for each project phase are identified in the Development Case. Deliverables are delivered towards the end of the iteration, as specified in section *4.1.3 Project Schedule*.

Deliverable	Description	Target Delivery Date
Project Plan	Initial project plan outlining objectives and scope	9/24
Public Repository	Create and share a public repository with group members	9/17
Finalized Project Plan	Completed and approved project plan	9/24
Software Requirements Document	Draft of functional and non-functional requirements	10/1
Finalized Software Requirements Document	Completed requirements document based on feedback	10/8
Project Architecture and Design Document	Initial architecture and design specifications	10/22
Finalized Project Architecture and Design Document	Completed architecture and design document	11/5
Implemented C++ Code	C++ code for the arithmetic expression evaluator	11/26
Project Test Cases	Draft of test cases based on requirements	11/26
User Manual/README	Instructions for using the evaluator	11/26
Finalized Test Cases	Completed test cases document	12/3
Finalized User Manual	Completed user manual with examples	12/3

Testing Results	Results from the testing phase	12/10
Final Project Submission	Complete project including all deliverables	12/10

2.5 Evolution of the Software Development Plan

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

Versions	Changes	Revision Criteria
1.0	Initial release of the Software Development Plan	Start of the project; includes initial objectives and scope
1.1	Updates based on the finalized project plan	Following the completion of the project plan on 9/24
1.2	Revisions based on finalized software requirements	After completing software requirements on 10/8
1.3	Updates based on finalized architecture and design documents	After completion of architecture and design on 11/5
1.4	Implementation updates and changes to the development approach	Following wrap-up of implementation on 11/26
1.5	Final revisions based on testing feedback	Post-testing phase, after finalizing testing on 12/10
2.0	Final version for project submission	Upon successful project completion and submission on 12/10

3. Project Organization

3.1 Organizational Structure

Project Manager - Responsible for overall project leadership, scheduling and planning meetings, resource allocation, ensuring the project is completed on time and meets specified requirements, primary interface with instructor.

Quality Control - Focuses on quality throughout the development process, resulting in higher-quality software.

Notetaker - Takes detailed notes meeting contents and action items, recording a log of team meetings.

Risk management - Identifies and addresses risks early in the project lifecycle, minimizing project delays

Configuration management - Maintains the version control of software artifacts, documents and tracks all changes made to the software's configuration

Requirements management - Responsible for gathering and documenting requirements, maintaining traceability, and managing changes by assessing their impact on project scope.

Implementer, tester, integrator - Every team member is responsible for contributing to the technical aspects of the project. Everyone will work together to develop the code, test the code, and properly integrate all requirements into the project.

Designer - Responsible for creating designs for the system

Developer - Responsible for writing code and integrating it with other components of the system.

Testers - Responsible for testing the code to ensure that it meets the specified requirements and is free of defects.

3.2 Roles and Responsibilities

Person	**Unified Process for EDUcation Role**
Emma Du	**Project manager** (designer, developer, tester) Contact info: du.emma@ku.edu, 913-777-8929
Kit	**Quality control** (designer, developer, tester) Contact info: kitmagar@ku.edu, 605-728-4031
Vrishank Kulkarni	**Notetaker** (designer, developer, tester) Contact info: vkulkar@ku.edu, 913-251-2800
Phoenix	**Risk management** (designer, developer, tester) Contact info: phoenixbrehm@ku.edu, 913-258-0706
Khang	**Configuration management** (designer, developer, tester) Contact info: k585p599@ku.edu, 316-295-8418
Jake	**Requirements management** (designer, developer, tester) Contact info: jakobhuffman00@ku.edu, 785-215-4136

https://www.upedu.org/process/workers/ovu_works.htm

Anyone on the project can perform Any Role activities.

4. Management Process

4.1 Project Plan

4.1.1 Iteration Objectives

Documentation Iteration Objectives:

- Project Plan
- Software Requirements
- Software Architecture Design
- Test Cases
- Users Manual

Software Iteration Objectives:

- Ability to parse through user input
- Operator support for addition, subtraction, multiplication, division, modulo, and exponentiation
- Interpret parentheses and establish precedence and grouping
- Support for numeric constants as integers, possible support for floating-point values

- User-friendly interface for ease of use
- Accounting for possible errors in input and resolving them

4.1.2 Releases

Software release versions will coincide with project part due dates

Project Plan (Demo)	A general plan for the project will be outlined in the Project Plan document for stakeholders
Software Requirements (Pre-Alpha)	Software requirements will be finalized to have a more concrete idea of how the project will be implemented
Software Architecture Design (Alpha)	More concrete work will be finished, with an early version of the product available
Test Cases (Beta)	The project will enter the testing stage for final touches
User Manual (Full Release)	Once the project is finished, a user manual will accompany it

4.1.3 Project Schedule

9/17 - 12/12

9/17	Start project plan, create public repository and share link with group members
9/24	Finalize project plan
10/1	Begin software requirements
10/8	Finalize software requirements
10/22	Start project architecture and design documents
11/5	Finalize project architecture and design
11/12	Start project implementation
11/19	Continue implementation
11/26	Wrap up implementation, begin project test case and user manual
12/3	Finalize project test case and user manual, perform testing
12/10	Finalize testing and submit project

4.2 Project Monitoring and Control

- Requirements Management:

  - We will keep a detailed log using a diagram which showcases dependencies from one part to another, this diagram will be maintained in a folder with all revisions in order to record the changes within our product requirements.
- Quality Control:
  - Using the due dates on canvas will provide a rough timeline in order to produce said project deliverables. Then what is discussed in team meetings will be what each member has to do by the end of the week, this information will be logged in the meetings log. The evaluation of parts of the project will be done by considering the descriptiveness of code comments, intended functionality of the code, and if applicable, performance of the code. If the work is considered inadequate by these standards, then the action to take will be discussed as a group by the next meeting.
- Risk Management:
  - Each individual team member should maintain frequent backups and frequent saving practice in the case of any kind of crash and/or data loss. Github will assist in the management of these backups. Team members should also maintain well thought out, and signed, code comments in order to make working together on coding tasks easier for any member.
- Configuration Management:
  - Changes shall be submitted via github, each team member should be responsible for reviewing one other team member's work under the standards set by the Quality Control section. Project Artifacts should be named in the format of [project-artifact-type_versionnumber_g15]. For example, project-plan_v1-01_g15 (g15 for group 15). Backup plan is recovering backups via github, the disaster plan is to have a team member keep an up-to-date version of the github repo on their local machine, as well as having the github backups,in the case that a machine is destroyed or if the repo faces a deletion issue, another team member can always re-upload the entire repo with minimal data loss. Team members should update their local files twice a week.

4.3 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.

4.4 Risk Management

Risks will be identified in Inception Phase using the steps identified in the RUP for Small Projects activity "Identify and Assess Risks". Project risk is evaluated at least once per iteration and documented in this table.

Refer to the Risk List Document (CCC-DDD-X.Y.doc) for detailed information.

4.5 Configuration Management

Appropriate tools will be selected which 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 one member of the project, the Change Control Manager role.

Refer to the Configuration Management Plan (EEE-FFF-X.Y.doc) for detailed information.

5. Annexes

Programming Guidelines

The code for the arithmetic expression parser should be written in C++ using object-oriented programming principles. The following guidelines apply:

- **Naming Conventions**: Use descriptive names for variables, classes, and functions (e.g. evaluateExpression).
- **Code Modularity**: Each function and class should be modular, performing a specific task to ensure ease of debugging and future extensibility.
- **Error Handling**: Implement exception handling for issues such as division by zero, invalid input, and unbalanced parentheses.
- **Code Comments**: Include comments to explain complex logic and function purposes.

Design Guidelines

The design of the arithmetic expression parser will adhere to these guidelines:

- **Data Structures**:
  - A stack-based approach or a binary tree will be used to represent and evaluate expressions.
  - Use a class structure to represent different components such as tokens (operators, operands), expressions, and evaluation logic.
- **Operator Precedence**:
  - Define operator precedence based on the PEMDAS rule, and ensure correct grouping using parentheses.
- **User Interface**:
  - Design a simple command-line interface that accepts user input and provides output or error messages based on the validity of the expression.

Process Guidelines

The development process for this project will follow these practices:

- **Version Control**: Use Git for version control to track changes, facilitate collaboration, and maintain backups.
- **Agile Workflow**: Organize work into sprints with specific goals, such as completing functions or testing the evaluation mechanism.
- **Code Review**: Conduct regular peer reviews to ensure adherence to guidelines and improve code quality.

Technical Standards

The project will adhere to the following technical standards:

- **C++ Standard**: Use the C++14 or later standard for modern language features and better performance.
- **Compiler Requirements**: Ensure compatibility with GCC or Clang compilers to allow cross-platform functionality.

  The project will follow the UPEDU process.

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