EECS 348: Software Engineering I

EECS 348 Group 5
Software Development Plan

Version <1.0>

Revision History

Date	Version	Description	Author
2/8/2026	1.0	Filled in 3.3	Lea
2/15/2026	1.1	Completed Sections 1, 2	Lea
2/15/2026	1.2	Completed Section 3	Zack
2/15/2026	1.3	Completed Section 4	Shashwat
2/19/2026	1.4	Completed Section 5	Jr

Table of Contents

Software Development Plan

1. Introduction

*[The introduction of the **Software Development Plan** provides an overview of the entire document. It includes the purpose, scope, definitions, acronyms, abbreviations, references, and overview of this **Software Development Plan**.]*

1.1 Purpose

The purpose of this Software Development Plan is to define the strategy, schedule, and resources for developing the Arithmetic Expression Evaluator. It serves as the primary guide for our group (Group 5) to manage the development lifecycle from requirements to final delivery.

1.2 Scope

This plan covers the design; implementation, testing, and documentation of a C++ based arithmetic expression evaluator. It includes the development of a parser, an evaluator engine, and a command-line interface.

1.3 Definitions, Acronyms, and Abbreviations

CLI: Command Line Interface

Expression Parsing: Our program should be able to parse arithmetic expressions entered by the user, considering operator precedence and parentheses.

Numeric Constants: Recognizes and calculates numeric constants within the expression.

Operator Support: Implements support for the following operators:

- - (subtraction)
- % (modulo)
- * (multiplication)
- ** (exponentiation)
- / (division)
- + (addition)

Parenthesis Handling: Ensures that the program can handle expressions enclosed within parenthesis to determine the order of evaluation.

PEMDAS: Parentheses, Exponents, Multiplication, Division, Addition, Subtraction.

Unary Operators: Support unary – and +

1.4 References

- External References
  - Project Description: Saiedian, Hossein. EECS 348: Term Project in C++ - Arithmetic Expression Evaluator. Spring 2026
- Internal Documents
  - Requirements Document
  - Design Document
  - Test Plan

- User Manual
- Vision Statement: "To build a reliable C++ program that evaluates math expressions using PEMDAS rules. We aim to demonstrate string software engineering skills by creation code that is clean, well-tested, and easy to understand."

1.5 Overview

This Software Development Plan contains the following information:

Project Overview —	Describes the purpose, scope, and objectives of the project, including the specific deliverables and the constraints.
Project Organization —	Details the organizational structure of the project, defining specific roles and responsibilities assigned to each team member.
Management Process —	Outlines the project estimates, schedule, and milestones. It also describes the mechanisms for monitoring progress, managing requirements, controlling quality, and handling the risks.
Applicable Plans and Guidelines —	Contains references to additional standards, such as specific C++ programming guidelines

2. Project Overview

2.1 Project Purpose, Scope, and Objectives

- Purpose: To develop an Arithmetic Expression Evaluator component for a larger compiler product being developed in C++.
- Objective: Develop a C++ program that parses and evaluates arithmetic expressions with support for nested parentheses and operator precedence.
- Scope: The system must handle integers and operators +, -, *, /, %, and **.

2.2 Assumptions and Constraints

- Assumption: Initial input will be integer constants only.
- Constraint: Must follow PEMDAS rules; Exponents are right-to-left associative.
- Constraint: Must be developed in C++ using object-oriented principles.

2.3 Project Deliverables

i) Project Plan – 02/22/2026

ii) Requirements Document – 03/15/2026

iii) Design Document – 04/05/2026

iv) Project Implementation – 05/07/2026

v) Project Test Cases – 05/07/2026

vi) User Manual / README – 05/07/2026

2.4 Evolution of the Software Development Plan

Version	Date	Milestone	Description
1.0	0/22/2026	Milestone 1	Initial release of the Project Plan.
1.1	03/01/2026	Milestone 2	Update following the complementation of Requirements & Design phases.

1.2	03/22/2026	Milestone 3	Revision prior to Alpha (Parser) implementation.
1.3	04/12/2026	Milestone 4	Revision prior to Beta (Evaluator) integration.
2.0	05/07/2026	Milestone 5	Final version reflects the completed project and lessons learned.

3. Project Organization

3.1 Organizational Structure

- The project team will all convene to decide on any changes or decisions regarding the project. Any disagreement about the project or the direction we decide to go will be decided fully by our project manager Lea. The work will eventually be reviewed and approved by our class TA.

3.2 External Interfaces

- There will not be any other external interfaces the project group will be interacting with regarding developing the project; however, we will be submitting the project to our class TA for approval and grading.

3.3 Roles and Responsibilities

MEETING TIME: 4:00 PM Sundays 30–60 min Weekly – Zoom Meetings

Person/Contact Info	**Unified Process for Education Role**
Alex (913) 575-0714	Requirements & Documents Lead
Lea (913) 742-0306	Project Manager
Shashwat (704) 804-6410	System Architect
Jr. (901) 208-7966	Front End Lead Developer
Zack (913) 522-3941	Back End Lead Developer
Alec (913) 200-1518	Quality Assurance

4. Management Process

4.1 Project Estimates

Task	**Hours**
Weekly Meetings (30 min x 6 weeks)	3 hours
Planning & Design	5 hours
Coding	15 hours
Testing	10 hours
Final Review	2 hours
Total	35 hours

4.2 Project Plan

4.2.1 Phase Plan

Phase	Tasks	Duration
Planning	Design structure, assign roles, define functions	Week 1-2
Coding	Each member codes their assigned topics	Week 3-5
Testing	Test all modules, fix bugs	Week 6
Final Review	Put everything together, final checks	May 7

4.2.2 Iteration Objectives

Iteration 1 (Week 1-2) — Planning & Design

- Define program structure
- Assign roles to each team member
- Complete design document
- Define all functions and how they connect

Iteration 2 (Week 3-4) — Coding

- Code all 9 topics (Variables, Operators, Conditions, Loops, Arrays, Strings, User Input, Pointers, Functions)
- Build the main menu
- Each member completes their assigned modules

Iteration 3 (Week 5) — Integration

- Combine all modules together into one program
- Make sure everything connects properly
- Fix any errors that come up

Iteration 4 (Week 6 - May 7) — Testing & Final

- Test all features of the program
- Fix any bugs found
- Final review and cleanup
- Submit by May 7

4.2.3 Releases

Release 1 — Demo Version (Week 5)

- Basic working version of the program
- Main menu is functional
- Most topics are coded and working
- Used to show progress to professor
- May still have some bugs

Release 2 — Beta Version (Week 6)

- All topics fully coded
- All modules connected together
- Testing is done
- Most bugs are fixed
- Almost ready for final submission

Release 3 — Final Version (May 7)

- Fully complete program
- All 9 topics working correctly
- No known bugs
- Ready for submission

4.2.4 Project Schedule

Project Schedule:

Phase	Tasks	Start	End
Planning	Design structure, assign roles	Week 1	Week 2
Coding	Code all 9 topics	Week 3	Week 5
Testing	Test all modules, fix bugs	Week 6	Week 6
Final Review	Final checks, submission	Week 7	May 7

Milestone Schedule:

Milestone	Target Date
Design Document Complete	End of Week 2
Coding Complete	End of Week 5
Demo Release	End of Week 5
Beta Release	End of Week 6
Testing Complete	End of Week 6
Final Submission	May 7

Weekly Meeting Schedule:

Meeting	When	Duration
Meeting 1	Week 1	60 mins
Meeting 2	Week 2	30 mins
Meeting 3	Week 3	45 mins
Meeting 4	Week 4	45 mins
Meeting 5	Week 5	45 mins
Meeting 6	Week 6	45 mins

4.2.5 Project Resourcing

Team Members & Roles:

Member	Role	Responsibilities
Shashwat	System Architect	Design structure, main menu, define functions
Alex	Developer	Code topics 1, 2, 3 (Variables, Operators, Conditions)
Jr	Developer	Code topics 4, 5, 6 (Loops, Arrays, Strings)
Zack	Developer/Tester	Code topics 7, 8, 9 (User Input, Pointers, Functions) + Testing

Skills Required:

Skill	Who Needs It
Basic C Programming	All members
Understanding of Functions	All members
Knowledge of Pointers	Zack
Testing & Debugging	Zack

Tools Required:

Tool	Purpose
VSCode	Writing code
GCC Compiler	Compiling C program
GitHub	Storing and sharing code
Google Docs	Writing documentation

Training Required:

Training	Who	Target Date

C Programming Basics	All members	End of Week 1
GitHub Usage	All members	End of Week 1
Pointers & Memory	Zack	End of Week 2

4.3 Project Monitoring and Control

[The following is a checklist of items to consider:

- *Requirements Management:*
  - *All requirements are based on the 9 C programming topics*
  - *Any changes to requirements must be agreed upon by all 4 team members*
  - *Changes are recorded and tracked during weekly meetings*
- *Quality Control:*
  - *All code must be reviewed before combining into final program*
  - *Any bugs found must be fixed before final release*
  - *Code must follow C programming standards from the course notes*

Check	***When***	***Who***
Code Review	*End of each iteration*	*All members*
Function Testing	*Week 6*	*Member 4*

- *Reporting and Measurement:*
  - *Progress is reported at every 30-minute weekly meeting*
  - *Schedule is updated if any delays occur*

Metric	***How Often***	***Who Reports***
Tasks completed	*Weekly*	*Project Manager*
Bugs found/fixed	*Weekly*	*Member 4*
Code progress	*Weekly*	*All Developers*

- *Risk Management:*

Risk	***Likelihood***	***Solution***
Member not completing task	*Medium*	*Redistribute work in team*
Code not working	*Medium*	*Debug together as a team*
Running out of time	*Low*	*Start early, follow schedule*
Member dropping out	*Low*	*Remaining members share the work*

- *Configuration Management:*

- *All code stored on* ***GitHub***
- *Each member works on their own branch*
- *Code is merged during Integration phase (Week 5)*
- *All documents saved on* ***Google Docs***
- *Final version submitted on* ***May 7***

4.4 Requirements Management

The requirements for this system are captured in the Vision document. Requested changes to requirements are captured in Change Requests and are approved as part of the Configuration Management process.

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.

4.6 Reporting and Measurement

Updated schedule estimates, and metrics summary reports, will be generated at the end of each iteration.

The Minimal Set of Metrics, as described in the RUP Guidelines: Metrics will be gathered on a weekly basis. These include:

Earned value for completed tasks. This is used to re-estimate the schedule and budget for the remainder of the project, and/or to identify need for scope changes.

Total defects open and closed – shown as a trend graph. This is used to help estimate the effort remaining to correct defects.

Acceptance test cases passing – shown as a trend graph. This is used to demonstrate progress to stakeholders.

Refer to the Project Measurements Document (AAA-BBB-X.Y.doc) for detailed information.

4.7 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.

4.8 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

Concerning programming guidelines, the following will be followed: Consistent coding style and naming conventions, error handling, input validation, operator handling, and documentation. For design guidelines, the project should use a tree data structure, containing nodes that hold either numeric values or arithmetic operators. Within design guidelines, there will be testing frameworks that evaluate incoming values for validity, reacting appropriately. For process guidelines, team members shall use Git to frequently commit changes with meaningful, clear messages to avoid ambiguity. Code pushes to the main branch will be reviewed by at least one other team member to confirm accuracy and efficiency. Documentation will be kept describing designs and the testing process; documentation will also be kept for milestones (such as implementing tokenizer, implementing parser, etc.). Errors within testing will be logged and reviewed by each team member.

The project will follow the UPEDU process.

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