348 Industries

EECS 348 Fall 2023 Term Project: Arithmetic Evaluation Program

Software Development Plan

Version <1.0>

Revision History

Date	Version	Description	Author
09/22/2023	1.0	First iteration of the software development plan for the program to be made for EECS 348 Fall 2023 Term project. This program will be able to execute various operations given valid inputs.	Dev Patel Jay Chung Ruth Higgason Colin Marett

Table of Contents

Software Development Plan

1. Introduction

This Software Development Plan outlines the strategic approach and framework for managing the development of the "Arithmetic Expression Evaluator in C++" project for EECS 348: Software Engineering in Fall 2023, under the guidance of Professor Hossein.

1.1 Purpose

The purpose of this Software Development Plan is to provide a comprehensive roadmap for the successful execution of the project. It serves as a primary reference for project managers, outlining the development approach and guiding principles. Additionally, it offers project team members a clear understanding of their roles, responsibilities, and dependencies.

1.2 Scope

This plan pertains to the management of the "Arithmetic Expression Evaluator in C++" project. It encompasses the entire project lifecycle, from inception to deployment. Specific details about individual project iterations will be documented in separate Iteration Plans. The contents of this plan align with the product requirements specified in the Vision Document.

1.3 Definitions, Acronyms, and Abbreviations

Please refer to the Project Glossary for definitions of project-specific terms, acronyms, and abbreviations.

1.4 References

The Software Development Plan references the following artifacts:

- Iteration Plans
- Development Case
- Vision Document
- Glossary
- Any other relevant supporting plans or documentation.

1.5 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.

These sections collectively guide the development effort for the "Arithmetic Evaluation Program in C++" project.

2. Project Overview

2.1 Project Purpose, Scope, and Objectives

The purpose of this project is to build a C++ program that, taking a string of a valid arithmetic expression as input, using the operators +, -, *, /, %, and ^ and numeric constants, can be parsed and evaluated. It can also handle grouping with parentheses.

The project is intended to have three deliverables which are, briefly:

1. Common software engineering artifacts
2. C++ Program operators and features.
3. README file for the program

2.2 Assumptions and Constraints

Staff: There are five group members available for this project

Equipment: Each member of the group has their own computer or has access to a computer at the school

Schedule: The project will be due by the end of the semester

2.3 Project Deliverables

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.

There are also other milestones in the project such as test cases, code, and design specs that are not yet well-defined and will be added once more is known about the project. The only deliverables that are expected are Common Software artifacts, C++ Program, operators and features and a README file.

2.4 Evolution of the Software Development Plan

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

3. Project Organization

3.1 Organizational Structure

The Team Administrator talks to the client and relays what the client wants to the team. The Product Management then breaks the project up into smaller parts and organizes when each part needs to be done. The Team Administrator and the Product Management both make sure that the project progresses on schedule to keep progressing in the project. The Project Lead 1 and Project Lead 2 both make sure that the project is coded correctly and all bugs are fixed. The Data Administrator/Quality Assurance tests the project throughout the process making suggestions of anything that should be or needs to be added to the project and also double checks for any bugs that need to be fixed.

3.2 External Interfaces: N/A

3.3 Roles and Responsibilities

Person	**Unified Process for EDUcation Role**
Chung, Jay	Product Management - Identifies requirements, manages and supervises features and programs. **Email - bluenone1025@gmail.com**
Higgason, Ruth	Project Lead 1 - Over looks the coding of the project, ensures that everything is correctly coded with no bugs. **Email - ruth.higgason@gmail.com**
Marett, Colin	Project Lead 2 - Looks the code part of the project, ensures bug fixes **Email - crmarett@gmail.com**
Patel, Dev	Team Administrator - Liaises with client, team and supervises. **Email - devpatel14343@gmail.com**
Reddy, Sanketh	Data Administrator/Quality Assurance - Testing the product to see if there are any bugs or features that should be added **Email - Sankethreddy77@gmail.com**

4. Management Process

4.1 Project Plan

Week 1 (09/11-09/17)	Team Organization, Role assignment, Project details analysis, Requirements engineering.
Week 2 (09/18-09/24)	Project plan conception, Implementation preparation, Team Meeting 1
Week 3 (09/25-10/01)	Requirement Specifications Planning, Phase 1 of implementation: Interface, Addition and Subtraction operators.
Week 4 (10/02-10/08)	Team Meeting, Phase 2: Multiplication, Division, Exponents
Week 5(10/09-10/15)	Team Meeting, Phase 3: Mixed Operators and Extraneous Parentheses
Week 6	Team Meeting, Project evaluation, Phase 4: Complex Addition and Extraneous Parentheses
Week 7	Team Meeting, Phase 5: Unary arithmetic, parentheses manipulation, bug fixes
Week 8	Team Meeting, Client evaluation, bug fixes

4.1.1 Iteration Objectives

- Phase 1: Here we will start with creating a basic interface for the user to navigate to each available operation and will continue to add to it as each iteration progresses. Then addition operations will be added, as well as subtraction operations. It will be reviewed before the next phase.
- Phase 2: Now we can implement the multiplication, division, and exponential operations including parentheses manipulation and outlier cases.
- Phase 3: Mixed operators, and extraneous parentheses will be added to the program, while making sure the previous code complies and does not have bugs.
- Phase 4: Complex operations with Extraneous parentheses will be implemented.
- Phase 5: Finally, Unary operations, more parenthese manipulation and bug fixes will be added.
- After the bugs have been fixes and user stories have been sorted, the product will be reviewed by the whole team and client.

4.1.2 Releases

There will be a release for each iteration phase of the software. The first iteration is 1.0 which will have the User interface and addition/subtraction operations. Then as each iteration passes, more is added.

4.2 Project Monitoring and Control

- Requirements Management: We can specify the information and control mechanisms which will be collected and used for measuring, reporting, and controlling changes to the product requirements.
    - Initial Requirement Specification: documenting the initial product requirements. This includes functional requirements (e.g., arithmetic operations, UI elements), non-functional requirements (e.g., performance, usability), and any constraints or assumptions.
    - Request Mechanism: Establish a formal change request mechanism that allows stakeholders to submit change requests when they identify a need for alterations to the requirements.
    - Request Evaluation: Team will decide collectively

  - Version Control System: Using a version control system to manage changes to the requirement documentation. Each change should be tracked, including who made the change and when.
  - Documentation: Retain all logs, documentation and files.
- Quality Control: Controlling the quality of project deliverables is essential for the successful completion of a software project. Quality control involves monitoring, evaluating, and taking corrective actions when necessary to ensure that project deliverables meet the specified requirements and standards.
  - Timing for Quality Control: Quality control should be performed throughout the project lifecycle, from the initial design phase to the final testing and deployment phase. Specific checkpoints and reviews should be scheduled at key milestones, such as after completing the software architecture, coding, and testing phases.
  - Walkthroughs: Early in the project, conduct walkthroughs where team members review the design and code to identify potential issues and discrepancies. This is more informal and focuses on understanding the project's direction.
  - Inspections: As the project progresses, conduct formal inspections to thoroughly examine the design, code, and other project artifacts. Inspections are more structured and systematic than walkthroughs.
  - Reviews: Regular peer reviews should be conducted during the coding phase. Code reviews involve team members examining the code for correctness, adherence to coding standards, and potential defects.

- Risk Management:
  1. Risk Identification:
     - Brainstorming: Begin by gathering project stakeholders, including developers, designers, and product owners, to identify potential risks. Encourage open and candid discussions to uncover various risk factors.
     - Documentation Review: Review project documentation, including requirements, design documents, and project plans, to identify risks related to scope, technology, and dependencies.
     - Historical Data: Use historical data from past projects to identify risks that are commonly associated with similar software development efforts.
     - External Inputs: Consider external factors such as market changes, regulatory changes, and vendor dependencies that could impact the project.
  2. Risk Analysis:
     - Qualitative Analysis: Assign a likelihood and impact rating to each identified risk. This helps prioritize risks based on their potential impact on project objectives.
     - Quantitative Analysis (if possible): For critical risks, consider performing quantitative analysis to estimate the potential cost and schedule impact more precisely.
     - Root Cause Analysis: Understand the underlying causes of each risk to develop effective mitigation strategies.
  3. Risk Prioritization:
     - Risk Categories: Categorize risks into different categories, such as technical, operational, or organizational, to better understand their nature and potential impact areas.
     - Risk Tolerance: Define risk tolerance levels with project stakeholders to determine which risks are acceptable and which require mitigation.

4. Risk Mitigation:
    - Risk Response Planning: Develop a risk response plan for each high-priority risk. Response strategies may include risk avoidance, risk reduction, risk transfer, or risk acceptance.
    - Contingency Planning: Create contingency plans for risks that cannot be fully mitigated. These plans outline how the project will respond if the risk materializes.
    - Monitoring Triggers: Define triggers that indicate when a risk is about to occur or has occurred. This allows for timely response.

- Configuration Management:

    1. Problem Submission: Team members and stakeholders can submit problems or change requests via a designated system or platform. Problems should include a clear description, severity level, and any relevant documentation or evidence. Problem Review: Problems are reviewed by a designated team, including developers, testers, and project managers. Severity and impact assessments are made to prioritize issues. Dispositioning: Issues are dispositioned as follows:
        - Accepted: Valid issues are assigned to the appropriate team for resolution.
        - Rejected: Non-valid or out-of-scope issues are rejected with clear reasons.
        - Deferred: Issues that can't be addressed immediately are scheduled for future releases.
    2. Conduct regular audits of the problem and change management process, naming conventions, and backup and disaster recovery plans to identify areas for improvement.
    3. Update and adapt these processes and plans as needed to address changing project requirements and technologies.

4.3 Quality Control

Defects will be recorded and tracked as Change Requests, and defect metrics will be gathered. All the 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

Identifying Risks:

- Team Brainstorming: At the initiation phase, the team members of the project together will conduct a brainstorming session to identify potential risks in the project. This includes technical, scheduling, management, quality, resources, and any other aspects.

- Continuous Monitoring: Risk identification is an ongoing process as we go deeper, we might find new risks. So, all team members will continuously monitor for any new risks during meetings, code reviews, and as we progress onto the next phase.

- Reviewing of Documentation: Team members will review documentation of the project and the different plans, so that we can identify any gaps, inconsistencies, or risks.

Analyzing Risks:

- Evaluate the potential impact of identified risks for this project.

Then categorize them as low, medium, and high impact based on their overall impact on this project.

To have a briefer idea on how we can label them, we can consider this:

Low Impact: Risks have minor consequences and are manageable.

Medium Impact: Risks have medium consequences and may require some more time and effort to manage them.

High Impact: Risks have higher consequences for this project if not addressed and rectified before jeopardizing the project.

- Evaluate the potential possibility of identified risks for this project together with the team members. Then categorize them as

Low Possibility: Unlikely or Low chances of occurring.

Medium Possibility: Moderate chances of occurring.

High Possibility: High chances of occurring.

Prioritizing Risks:

- We can prioritize risks together as a team based on what we find while evaluating the impact and possibility of risks from analyzing risks.

Monitoring Risks:

- Keeping a log of the risks identified as Risk List Document.
- Regular review session of the risks during meetings.
- Assessing the risks during the meetings.
- Updating the status of the risks.
- Updating any new risks identified to the document.

Mitigating Risks:

- Develop a specific mitigation plan for the risks identified.
- Implement a plan of action to reduce the possibility or the impact of the risk.
- Have a contingency plan for identified risks.
- Detect the problems of the risk early and have a solution for it.
- To avoid technical risks, we need to test our code at every possibility before they become more critical.

4.5 Configuration Management

1. Problems and Change Requests

- Any team member who identifies a problem or suggests a change can let the team members know.
- The team members will review the problems and changes and act accordingly.
- Approved changes by the team will be implemented by the team member who is responsible for the relevant changes, keeping in mind of the plan and schedule.

2. Artifact Naming, Marking, and Numbering

- The artifacts should be named in a clear way that conveys a meaning by their brief name and they should be named in a consistent manner.
- Important documents and code may include headers and comments describing the purpose, author, version, date, etc.

- The artifacts should be assigned a version number so that we can differentiate between different versions. Ex: Code1.0. Code1.2. Code2.0.
- Use a version control system (ex: git) to track different versions and changes to the code and documentation.

3. Retention Policies

- All the project's artifacts including the documentation, code, and plans should be retained for the rest of duration of the project.
- After the project completion and after it's graded it can be achieved for future reference if needed.

4. Back – Up Plans

- Regularly back up the project's artifacts.
- Store backups in a secure location, either on local device, external hard drives or on cloud services.
- It is suggested to keep multiple copies of the artifacts in different locations.

5. Disaster Plans

- It is suggested for all the team members to keep a copy of the artifact, in case if someone loses it other team members can provide a backup for the lost artifact.

6. Recovery Plans

- Artifacts can be recovered from the different locations they were saved in.
- Artifacts can be recovered by asking fellow team members if they have a copy of the artifact.

5. Annexes

- The project will follow the UPEDU process.
- Programming Guidelines: The programming language used in this project is C++.
- Design Guidelines: They will define the design principles, patterns, and structure of the software components and other artifacts. They will provide consistency and maintainability throughout all the components and artifacts.
- Test Plan: Provides a detailed outline of the testing approach.
- Risk Management Plan: Outlines the approach of identifying, analyzing, prioritizing monitoring, and mitigating risks throughout the project.
- Configuration Management Plan: Provides an overview of the project's configuration management practices throughout the project.