Quick Math.io

Arithmetic Analyzer
Software Development Plan
Version 0.2

Revision History

Date	Version	Description	Author

17/09/23	0.1	Added notes and filled descriptions	Vinny, Omar, David, Owen, Tatum, Jamie
21/09/23	0.2	Finalized some sections	David, Tatum

Table of Contents

Software Development Plan

1. Introduction

1.1 Purpose

The purpose of the Software Development Plan is to ensure that all of the needed information to complete the project is included within one space. It discusses the approach used to complete the project and addresses the project management plan. The approach to the development of the software is to use the Shunting-Yard Algorithm and Pushdown Automata to achieve the code output portion of the project.

The following people use the Software Development Plan:

- The Project Manager: The project manager uses the project plan to initiate, execute, monitor, and close a project. They use it to outline objectives, tasks, and timelines to ensure the successful completion of the project.

- The Deputy Administrator: The Deputy Administrator assists the project manager in organizing tasks, tracking progress, and ensuring that the project objectives are met within the established timelines.

- The Quality Assurance Engineer: The quality assurance engineer uses the project plan to establish and enforce quality standards and procedures, ensuring that the project's deliverables meet the specified quality criteria and standards throughout the project's lifecycle.

- The Configuration Management Engineer: The Configuration Management Engineer uses the project plan to oversee the management of project artifacts, ensuring version control, documentation integrity, and proper configuration of all project-related components to maintain consistency and traceability throughout the project's development.

- The Lead Software Engineer: The Lead Software Engineer uses the project plan to guide the technical development of the software, allocate tasks to the development team, ensure adherence to coding standards, and coordinate efforts to deliver high-quality software solutions in alignment with project goals and timelines.

- Meeting Coordinator: The meeting coordinator uses the project plan to schedule and facilitate project-related meetings, ensuring that they are aligned with project milestones and objectives.

1.2 Scope

This Software Development Plan encompasses the planning, execution, and management of the specific software development project for team Quick Math.io. This document outlines the project's objectives, deliverables, timeline, resources, roles, responsibilities, and the overall project approach. It influences and guides the activities of the project team, ensuring that all project stakeholders have a clear understanding of the project's goals and how they will be achieved.

1.3 Definitions, Acronyms, and Abbreviations

See the Project Glossary.

1.4 References

https://en.wikipedia.org/wiki/Shunting_yard_algorithm

https://en.wikipedia.org/wiki/Pushdown_automaton

https://en.wikipedia.org/wiki/Stack_(abstract_data_type)

https://en.wikipedia.org/wiki/Parsing

1.5 Overview

The subsequent section outlines the content and structure of this Software Development Plan. It includes the following components:

- Project Overview: This section provides insight into the project's purpose, scope, objectives, and expected deliverables.
- Project Organization: This section includes the organizational structure of the project team.
- Management Process: This part outlines the estimated cost, schedule, major project phases, milestones, and the project's monitoring approach.
- Applicable Plans and Guidelines: This section offers an overview of the software development process, encompassing the methods, tools, and techniques to be employed throughout the project.

2. Project Overview

2.1 Project Purpose, Scope, and Objectives

Purpose: To create a C++ program that can parse and evaluate arithmetic expressions with various operators and parentheses.

Scope: The program will handle integer expressions with operators +, -, *, /, %, ^, and parentheses for grouping and precedence.

Objectives: Parse input expressions, evaluate expressions while obeying the order of operations, handle parentheses, recognize numeric constants, provide a user interface, implement error handling.

2.2 Assumptions and Constraints

Assumptions: users will provide valid integer expressions as input. Only the specified operators will be supported.

Constraints: timeline for delivery of the project. Staffing resources for design, coding, and testing.

2.3 Project Deliverables

Requirements documents

Design Document

Test cases

C++ code and executable program

User Manual

Project management deliverables (project plan, etc.)

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	Changes
0.1	*09/17/2023*	*Initial version*
0.1	*09/21/2023*	*Finalizing sections*

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

3. Project Organization

3.1 Organizational Structure

The organizational structure is defined by six individual roles for each team member to focus on. The project administrator is tasked with managing the project, providing leadership, and acting as the primary representative for the group when interacting with the instructor or client. The deputy administrator is selected by the project administrator to assist in these duties. The quality assurance engineer focuses on the product's operating

quality. This includes generating and implementing tests for functionality, usability, and performance. The configuration management engineer sets up and maintains build/deployment systems, tracks changes, manages branching and merging strategies, and ensures consistent environments across development. The meeting coordinator documents details of team meetings, manages the team schedule for meetings, and coordinates with the project administrator to ensure the project is following the necessary progress timeline. All members will interchangeably contribute to the development of the software solution; However, the lead software engineer takes command of programming-related operations.

3.2 Roles and Responsibilities

Person	Unified Process for EDUcation Role
David Levy	Project Administrator
Tatum Morris	Deputy Administrator
Jamie King	Quality Assurance Engineer
Vinayak Jha	Configuration Management Engineer
Omar Faruque	Meeting Coordinator
Owen Krussow	Lead Software Engineer

Anyone on the project can perform Any Role activities.

4. Management Process

4.1 Project Plan

We plan to use the shunting-yard algorithm to convert an infix expression to a postfix expression, so we can parse the expression using some sort of pushdown automaton mechanism.

Once that has been done, we will implement features in the following order and for the following reasons:

1. Addition; for the purpose of adding two numbers.

2. Subtraction; for the purpose of subtracting two numbers.

3. Parentheses; for the purpose of organizing operations.

4. Multiplication; for the purpose of multiplying, or repeated addition between, two numbers.

5. Division; for the purpose of dividing two numbers as the inverse operation to multiplication.

6. Exponentiation; for the purpose of exponentiating, or repeated multiplication between, two numbers, i.e. a base and an exponent.

7. Unary sign operators; for the purpose of defining magnitudes of scalar values.

8. Error throwing; to prevent the user from attempting to divide by 0, using an odd number of parentheses, creating syntactically wrong expressions, using incorrect characters or operators, or attempting to misuse the calculator in any other way.

4.1.2 Iteration Objectives

(Not sure where these documents are exactly, but I'll put this for right now):

- The first step we will take is to create a script that takes an infix expression as input and returns a postfix expression.
    - To do this, we will implement the shunting-yard algorithm.
- Next, we will write a script that evaluates an infix expression, outputting the final answer.
    - This will involve creating our own pushdown automata implementation
    - The post-fix expression will be pushed onto our pushdown automata implementation.
    - Then iteratively pop from our stack to evaluate the expression
- Then we will write a wrapper/interface that the user can interact with to evaluate mathematical expressions.
    - To start, a simple command line interface
    - Later we intend to make a graphical interface

4.1.3 Releases

- Release #1 (beta):
    - Shunting-Yard algorithm implementation completed, which is used to convert infix to postfix expression.
    - Stack implementation completed, which is used to convert postfix expression into solution.
- Release #2 (beta):
    - Support for expressions, via evaluation of the postfix expression to produce a definitive answer.
- Release #3 (demo):
    - Correct edge cases that were created from Release #2.
- Release #4 (stable):
    - Create a streamlined user interface to foster efficiency between client and software.

4.1.4 Project Schedule

1	*Implement the Shunting-Yard Algorithm*

1a	*Verify correctness*
2	*Implement the Pushdown Automata*
2a	*Verify correctness*
3	*Create conditions to support arithmetic*
3a	*Verify correctness*
4	*Correct edge-cases*
4a	*Verify correctness*
5	*Create user interface*
5a	*Verify correctness*

4.2 Project Monitoring and Control

This section covers how we plan to monitor and control the project.

4.2.1 Quality Control

Quality will be monitored by using test cases to verify correctness during the development period. During the planning phase, the Quality Assurance engineer will be scrutinous and will verify that the plan is feasible.

Any inadequacies will be addressed on the spot and a GitHub ticket will be created by the Quality Assurance Engineer. These inadequacies will be addressed by developers and then checked by the quality assurance engineer. After corrective action is taken, the ticket will be resolved, and the issue will be removed from the ticket list.

All artifacts will go through a review process (by the team) before being submitted for release. The review process will include the aforementioned test cases, where all team members will test the system using 1-10 personal tests.

4.2.2 Risk Management

Risks will be identified through a review process carried out by team members. If ever a risk is identified, a team member will notify the Quality Assurance Engineer, and make a ticket. It will then be the quality assurance engineer's duty to carry out the fix. The review process will occur once per release (project deliverable) unless further review is necessary.

The team has identified the following risks for now (organized with the risk being N.a, and the solution being N.b)

1.a) A team member drops the class

1.b) In the event that this happens, the Project Administrator will evenly distribute tasks that were originally delegated to the (now-gone) member.

2.a) A team member becomes ill in some capacity

2.b) In the event that this happens, the team will periodically check in with the ill member. The Project Administrator will work with the ill member to reduce their workload if necessary.

3.a) A team member is unable to finish assigned tasks before a deadline

3.b) In the event that this happens, the Project Administrator will work with the team member to come to some solution. If there are repeated offenses, the Project Administrator will reach out to their lab TA for support in the matter.

4.a) Two team members get into an altercation

4.b) In the event that this happens, the Project Administrator will separate the members such that their work does not overlap. No snide comments will be tolerated.

If any of these problems occur and the Project Administrator is involved, the Deputy Administrator will act in the position of the Project Administrator. For example, if the Project Administrator drops the class, the Deputy Administrator will redistribute tasks.

4.2.3 Configuration Management

Configurations and artifacts will be managed and stored using git and Github.com. Tickets will also be managed via git and Github.com. Whenever a ticket is created, it will be administered and monitored using their service. The Quality Assurance Engineer and Change Control Manager will work together in this domain.

Program configurations will be updated whenever a significant change is made to the source code. When updated, program comments and documentation will also be updated. All changes will be verified and permitted by the Change Control Manager.

All additions to artifacts and executables will be included in every release.

5. Annexes

Programming Guidelines:

1. Configuring linters to have a standard code style across the team. (TODO: Research into C++/C linters)

2. Each function should do just one task and should be as small as possible.

3. Every file should be less than 300 lines of code.

4. Add unit tests for complicated functions.

5. No more than two nested levels of branching depth.

6. Code should either be completely memory leak-free, or the memory leak be reasonably bound to less than one-eighth of available memory.

Design Guidelines:

1. Avoid monolithic code design.

2. The general flow will start with a parser, then validation of syntax, then arithmetic analysis, and finally delivery of the result. Each step should be completely isolated from others, and the components should interact with interfaces.

This is done to encourage good code design and ease of swapping one implementation of a component from another. Additionally, such a code design allows for simplicity in writing unit tests.

3. The entire app should be completely containerized, with an official Docker image provided. This is done to foster easy deployment.

GitHub relevant guidelines:

1. Every change to the main branch should be made via pull requests, with at least 2 reviewers, and all approvals before being merged into the main branch.

2. The commit message should be less than 50 characters, with additional description if needed.

3. Use GitHub milestones to track major milestones.

4. Document bugs and issues using GitHub issues and make pull requests to resolve those issues.

GitHub Automatic Tests (maybe configure this?)