Foobar Inc

C++ Arithmetic Evaluator
Software Architecture Document

Version 0.3

Revision History

Date	Version	Description	Author
010/Nov/2023	0.1	Initial Project Architecture	Tuan Vu
12/Nov/2023	0.2	Detailed Project Architecture	Kyle Moore
12/Nov/2023	0.3	Cleanup formatting, fix concerns brought up during November 12th meeting. • Address OOP Architecture	Cody Duong

Table of Contents

Software Architecture Document

1. Introduction

1.1 Purpose

The Software Architecture Document (**SAD**) is a guide for **the project**. It breaks down **the project**'s structure, design principles, and important considerations.

This document includes an introduction, what software architecture the arithmetic evaluator uses, goals and constraints of the project related to the architecture, subsystems, the interface, and the overall Quality.

1.2 Scope

The software architecture document is a concise guide for developers, project managers, and stakeholders. It outlines high-level design, fostering effective communication, informed decision-making, and streamlined collaboration throughout the software development process.

1.3 Definitions, Acronyms, and Abbreviations

The project/this project/the program: "C++ Arithmetic Evaluator", described by the Software Development Plan (**SDP**). The repository for all code and documentation lives on GitHub, at https://github.com/codyduong/EECS-328-Project/.
SDP: Software Development Plan. See *Section 1.4 References — SDP*
SAD: Software Architecture Document. See *Section 1.4 References — SAD*
SRS: Software Requirements Specification. See *Section 1.4 References — SRS*

1.4 References

SDP: Available at https://github.com/codyduong/EECS-328-Project/
SAD: Available at https://github.com/codyduong/EECS-328-Project/
SRS: Available at https://github.com/codyduong/EECS-328-Project/

1.5 Overview

The document is organized into sections, The subsequent sections cover Architectural Representation, Architectural Goals and Constraints, Use-Case View, Use-Case Realizations, Logical View, Interface Description, Size and Performance, and Quality. Each section contributes to a holistic understanding of the project's architecture and design decisions, catering to different audience's needs.

2. Architectural Representation

The software architecture of the C++ Arithmetic Evaluator is represented through a traditional software architecture that includes a main program using various Object Orientated Programming modules and classes. Model elements encompass a console line for user interaction, a parser managing input, and a queue executing arithmetic operations. In the **physical view**, these components are encapsulated in an executable or module. The **logical view** delineates the user's perspective, highlighting the console, parser, and queue. The **process view** elucidates dynamic interactions, emphasizing the main program orchestrating processes. The **development view** details source code modules and dependencies as in the parser queue and command line interface.

3. Architectural Goals and Constraints

The architectural goals for the C++ Arithmetic Evaluator include:

- Safety: Ensuring that the evaluator handles input and expressions safely, preventing runtime errors or crashes.

- Security: Implementing secure coding practices to protect against potential vulnerabilities.
- Portability: Designing the system to be easily portable across different platforms and environments.
- Development Tools: Utilizing Google Test, clang++, cmake, and git for code development and testing.
- Team Structure: Coordinating development efforts among the team leader, quality assurance engineers, and configuration engineers, to ensure collaboration and efficiency.

4. Logical View

4.1 Overview

This subsection describes the overall decomposition of the design model in terms of its package hierarchy and layers.

4.2 Architecturally Significant Design Modules or Packages

4.2.1 Presentation Layer

User Interface Class:

- Accepts user input and output.
- Includes error handling.
- Dependent on Parser class.

4.2.2 Business Logic Layer:

Abstract Syntax Node Class:

- A class which handles representation of arithmetic operations between two operands and one operator, providing a result.

Expression Parser Class:

- A queue that deals with each of the arithmetic operators +, -, *, / and parenthesis
- Dependent on Abstract Syntax Node Class.

Parser Class:

- Correctly enqueues user input into Expression Handling Class.
- Dependent on Expression Parser Class.

5. Interface Description

- **User Input:**
- The command line interface accepts mathematical equations from the user as text input.
- Users can enter equations containing basic arithmetic operations (+, -, *, /), parentheses, and numerical values.
- Examples of valid input:
  - **2 + 3 * (4 - 1)**
  - **(2 + 4) / 6**
- Examples of invalid input:
  - **2 ++ 3**
  - **((2 + 4)**
- **Command Line Prompt:**
- The interface displays a command line prompt to signal that the system is ready to receive input.
- Will show an error message and ask for valid input again if the input is invalid.

6. Quality

- **Extensibility:** The software architecture is designed to support easy extensibility, allowing the addition of new features and functionalities without significant modifications to existing code. Well-defined interfaces and separation of concerns facilitate the integration of new functionalities with minimal impact.

- **Reliability:** The architecture prioritizes reliability to ensure consistent and accurate results under varying conditions. This includes error handling. Error handling mechanisms are implemented to gracefully manage unexpected situations, providing informative feedback to users. Robust validation processes are in place to catch potential issues with user input, preventing calculation errors and enhancing the overall reliability of the system.

- **Maintainability:** The architecture prioritizes maintainability to simplify ongoing development, debugging and updates. Codebase is well-organized and follows established coding standards, making it easier for developers to understand and contribute to the system. Documentation is comprehensive, providing clear insights into the architecture, design decisions, and code structure. Dependency management and version control practices are in place to streamline updates and reduce the risk of introducing regressions.

- **Usability:** The architecture considers usability aspects to provide an intuitive and user-friendly experience. The console-based interface is designed with clarity and simplicity, ensuring users can easily input equations and interpret results. Feedback messages are informative and user-friendly, guiding users in correcting errors and understanding the system's behavior.