<Calculator >

Version <1.0>

[Note: The following template is provided for use with the Unified Process for EDUcation. Text enclosed in square brackets and displayed in blue italics (style=InfoBlue) is included to provide guidance to the author and should be deleted before publishing the document. A paragraph entered following this style will automatically be set to normal (style=Body Text).]

[To customize automatic fields in Microsoft Word (which display a gray background when selected), select File>Properties and replace the Title, Subject and Company fields with the appropriate information for this document. After closing the dialog, automatic fields may be updated throughout the document by selecting Edit>Select All (or Ctrl-A) and pressing F9, or simply click on the field and press F9. This must be done separately for Headers and Footers. Alt-F9 will toggle between displaying the field names and the field contents. See Word help for more information on working with fields.] Marked (shaded) areas: items

that are OK to leave out.

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| <30/10/2003> | <1.0> | First additions | Xavier Ruyle |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Table of Contents

1. Introduction 4

1.1 Purpose 4

1.2 Scope 4

1.3 Definitions, Acronyms, and Abbreviations 4

1.4 References 4

1.5 Overview 4

2. Architectural Representation 4

3. Architectural Goals and Constraints 4

4. Use-Case View 4

4.1 Use-Case Realizations 5

5. Logical View 5

5.1 Overview 5

5.2 Architecturally Significant Design Packages 5

6. Interface Description 5

7. Size and Performance 5

8. Quality 5

# Introduction

[The introduction of the **Software Architecture Document** provides an overview of the entire **Software Architecture Document**. It includes the purpose, scope, definitions, acronyms, abbreviations, references, and overview of the **Software Architecture Document**.]

## Purpose

[This document provides a comprehensive architectural overview of the system, using a number of different architectural views to depict different aspects of the system. It is intended to capture and convey the significant architectural decisions which have been made on the system.]

[This section defines the role or purpose of the **Software Architecture Document**, in the overall project documentation, and briefly describes the structure of the document. The specific audiences for the document are identified, with an indication of how they are expected to use the document.]

The purpose of this document is to convey the design of the calculator project in regards to the actual implementation. It will specify the details and design of each class and how they relate to meeting the requirements needed for the calculator

## Scope

[A brief description of what the Software Architecture Document applies to; what is affected or influenced by this document.]

The Software Architecture Document is related to the design and structure of the calculator project as a whole.

## Definitions, Acronyms, and Abbreviations

[This subsection provides the definitions of all terms, acronyms, and abbreviations required to properly interpret the **Software Architecture Document**.  This information may be provided by reference to the project’s Glossary.]

UI: User Interface

IFN: Infix Notation

PN: Postfix Notation

EXPR: Expression

## References

[This subsection provides a complete list of all documents referenced elsewhere in the **Software Architecture Document**. Identify each document by title, report number (if applicable), date, and publishing organization. Specify the sources from which the references can be obtained. This information may be provided by reference to an appendix or to another document.]

N/A – no references at this moment

## Overview

[This subsection describes what the rest of the **Software Architecture Document** contains and explains how the **Software Architecture Document** is organized.]

The rest of the Software Architecture document will detail the architectural representation as well as its goals and constraints. It will then describe the use cases and the subsystems that will be used.

# Architectural Representation

[This section describes what software architecture is for the current system, and how it is represented. It enumerates the views that are necessary, and for each view, explains what types of model elements it contains.]

# Architectural Goals and Constraints

[This section describes the software requirements and objectives that have some significant impact on the architecture; for example, safety, security, privacy, use of an off-the-shelf product, portability, distribution, and reuse. It also captures the special constraints that may apply: design and implementation strategy, development tools, team structure, schedule, legacy code, and so on.]

# Use-Case View

[This section lists use cases or scenarios from the use-case model if they represent some significant, central functionality of the final system, or if they have a large architectural coverage—they exercise many architectural elements or if they stress or illustrate a specific, delicate point of the architecture.]

## Use-Case Realizations

[This section illustrates how the software actually works by giving a few selected use-case (or scenario) realizations, and explains how the various design model elements contribute to their functionality. If a Use-Case Realization Document is available, refer to it in this section.]

# Logical View

[This section describes the architecturally significant parts of the design model, such as its decomposition into subsystems and packages. And for each significant package, its decomposition into classes and class utilities. You should introduce architecturally significant classes and describe their responsibilities, as well as a few very important relationships, operations, and attributes.]

## Overview

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

The calculator will utilize a tokenizer to divide user inputted mathematical expressions into individual token strings which will then be sent to a queue. It will then send the tokenizer result to a parser which will use an algorithm to convert the tokenizer queue to PN. The PN will be stored in a queue. The PN will then be calculated using a basic algorithm using a stack using the calculator class. Finally, a display manager class will output the final result obtained from the calculator class.

## Architecturally Significant Design Modules or Packages

[For each significant package, include a subsection with its name, its brief description, and a diagram with all significant classes and packages contained within the package.

For each significant class in the package, include its name, brief description, and, optionally, a description of some of its major responsibilities, operations, and attributes.]

Calculator Project

* Tokenizer
  + **Description**: The tokenizer will tokenize a given input string from the user into the expressionQueue. Given an input from the user, it will translate each part of that input into meaningful ints, floating points, operators, represented by a string. It will also smartly strip whitespace.
  + Decimals will not signal a new token must be made. Instead, the tokenizer will know that it is a floating point
  + EX: "2+ 3.0004 \* 2" - > ["2","+","3.0004","\*","2"]
* Parser
  + **Description**: Parses a given expression queue that has been tokenized by the tokenizer. Uses an algorithm to convert the infix notation (expressed passed by user input) to postfix notation (a notation which is easier to calculate with). The infixToPostfix algorithm uses a queue and a stack to temporarily place operators in the operatorStack and place them back in the outputQueue when needed to facilitate PEMDAS requirements. The outputQueue is used for future use for the calculator class. For precedence, the parser uses a map which corresponds to PEMDAS rules
* Calculator
  + **Description:** Calculates a given postfix expression which is obtained from the parser. Result is stored in a final result attribute.
  + Uses a stack to calculate the PN EXPR.
* Display Manager
  + **Description:** The display manager class will provide an interface for the user to input data to the project and expect an output corresponding to that input. It will work with the calculator class to output a final answer for any input given by the user.

# Interface Description

[A description of the major entity interfaces, including screen formats, valid inputs, and resulting outputs. If a User-Interface Prototype Document is available, refer to it in this section]

The interface that the user will interact with is the command line. The user will be able to input mathematical expressions which can handle the following operators, +, -, \*, /, %, ^ or \*\*. Expressions must be mathematically valid or else there will be errors.

The command line will show the following

Input Expression: 3+4 *(user input)*

Answer: 7

Input Expression: (((((5 - 3))) \* (((2 + 1))) + ((2 \*

3)))

Answer: 12

# Size and Performance

[A description of the major dimensioning characteristics of the software that impact the architecture, as well as the target performance constraints.]

# Quality

[A description of how the software architecture contributes to all capabilities (other than functionality) of the system: extensibility, reliability, portability, and so on. If these characteristics have special significance, such as safety, security or privacy implications, they must be clearly delineated.]

The calculator is designed to ensure that the classes address various quality attributes.

In regards to extensibility each class is built to have its own purpose. For instance, the calculator can add new operators if the project were to be expanded. The parser could be changed to address a change in operator precedence. And the tokenizer could change to address other forms of multiplication (e.g 5(2+3) = 5\*(2+3)). The calculator is reliable as the project accounts for bad user input and has multiple different types of errors which will inform the user what went wrong. The project also accounts for portability as the command line is hardware agnostic.