# Introduction to Software Development

## Group Report: Calculator

# Table of Contents

## Overview

An overview of the studied problem, including a mind map. Comments about the adopted Software Life-Cycle Development (SLCD) as well as reflections about the project management process within the team, using the SWOT analysis and providing a work breakdown structure and a Gantt chart.

## The Approach

A description of your adopted approach (techniques and assumptions) and your adopted design, in particular using Data Flow Diagrams (DFD) and Finite-State Machine (FSM) techniques.

## Requirements

Detailed Requirements of the proposed version 2 as well as the analysis of the specifications and functionalities (in particular, do the MOSCoW analysis of requirements and risk analysis) and justifications on how the version 2 is related/improves the version 1 of the software.

## The Process

Explanations about the software implementation, parameters, and adopted software testing processes and metrics.

## The Results

A discussion about your results (reflection of testing approach, reflection on performance, such as time, accuracy, etc.).

## Conclusions

Conclusions (reflection on the adopted method and alternatives, reflection on the development – what went right/wrong, reflection on ethics, reflection on possible improvements).

## Code and Documentation

Debugged source code in C++, should be structured and commented. The code should show extensions as different versions of the software.

## Appendix

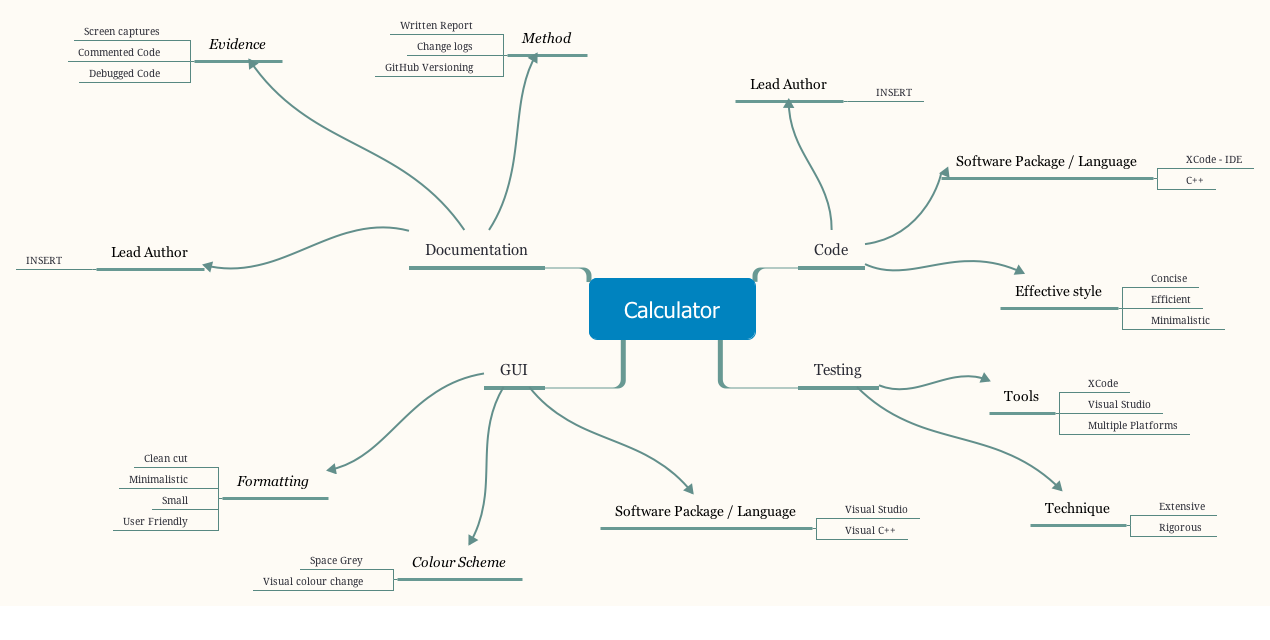
All files, evidence, results.

# Overview

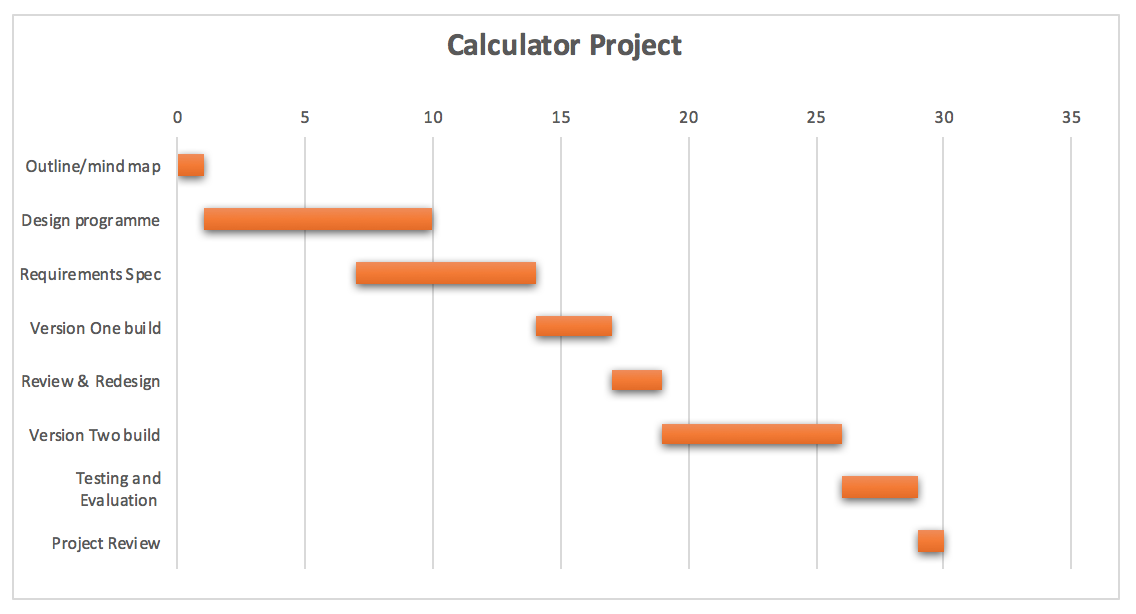
The objective of this assignment is to design and develop a complex calculator using C++, taking an input, which could be any combination of numbers (1-9) and operators (+, -, \*, /) in order to provide a correctly calculated output that can be clearly displayed to a user. In keeping with effective methods of project management the task should be approached by developing the software in incremental iterations. The first iteration should solve basic calculations (e.g. 2+2, 4\*8). The second should extend the calculator to facilitate solving unary calculations such as, inverse, power, and square root. The third iteration of development should implement the transition from a command line based software to the inclusion of a Graphical User Interface (GUI). Teams are also expected to incorporate a memory function that enables the storage and retrieval of the last calculation. The final iteration should extend the calculator to handle multiple operator precedence. Suitable tests will be executed to evidence the functionality and capability of the software to ensure the requirements and specifications are competently met.

## Mind Map

A mind map was created to initiate the design process, an important step that through collection of ideas assisted the team in understanding the scope of the project, the necessary tools and a shared image of the final product.

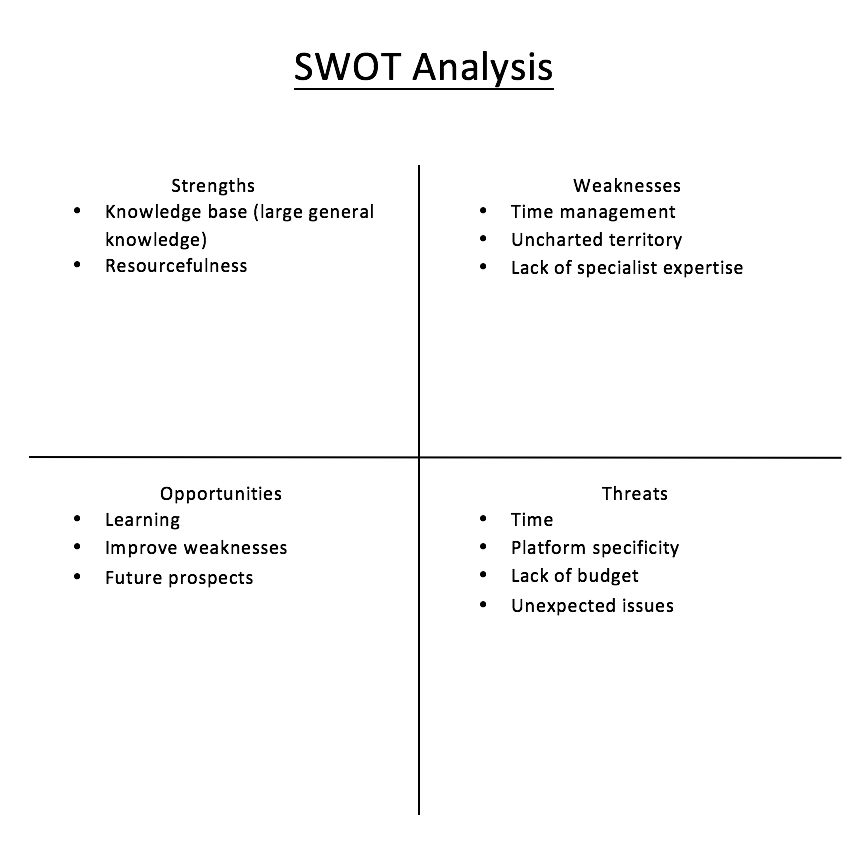


## Gantt Chart

A Gantt chart was created to organise the entire process by defining time scales and assigning tasks to members of the team.

## SWOT Analysis

A SWOT analysis was created to determine the strengths, weaknesses, opportunities, and threats involved in the process of developing a complex calculator.



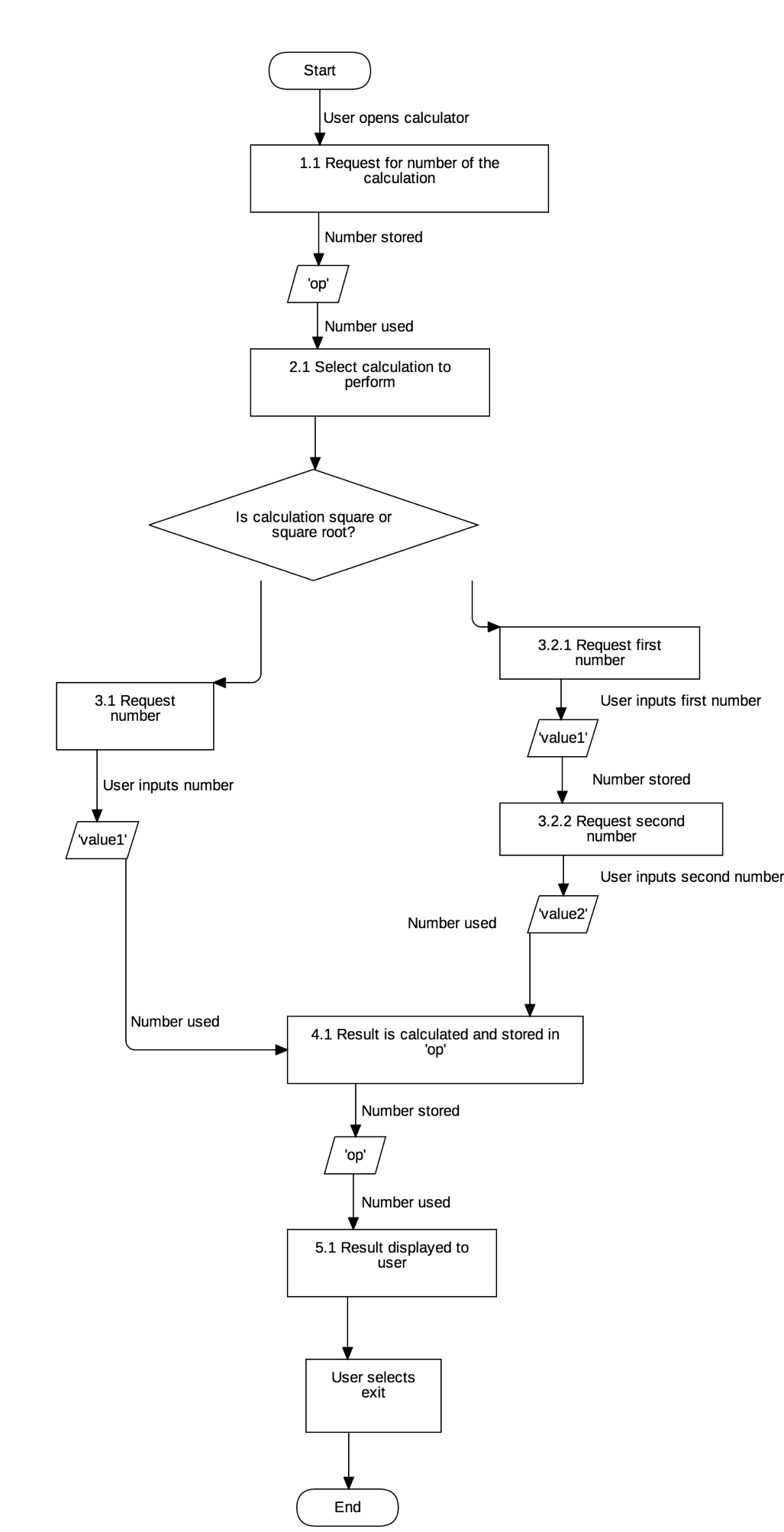
# The Approach

In approaching the task, the team decided it would be most beneficial to use Data Flow Diagram (DFD) and Finite-State Machine (FSM) techniques. It was assumed that users of the developed software would have a basic knowledge of the usability, functionality and general capabilities executable via both standard, scientific, analog and digital calculators. It was also assumed that users understand the principles of the prominent operators used within the software and how to interact with them. For example, it is understood that users will be able to follow basic intuitive interfaces and follow on screen instructions in English in order to use the calculator.

The first task was to design the various stages of the calculator from a user’s point of view, from this the team was able to more effectively develop the code to meet the primary needs of a generic calculator user. Through creation of a DFD and FSM, code implementation became much more efficient.

The initial version of the calculator was compiled and executed using the terminal command line. This helped with quickly running the software to check partial changes that were made along the creation process. The calculator was established by using a simple switch statement. This began by including the declaration of relevant libraries, in this case ‘iostream’ which facilitated the definition of standard input and output stream objects. The program utilized a standard namespace. A main function was created, an integer with a return value of zero. Then variables were defined, one to hold operators: ‘op’, and floated ‘a’ and ‘b’ variables to hold the two number values. In the relevant sections messages were implemented to guide the user through the process of operating the calculator. For example, a welcome message is presented with a list operator functions. The user selects which they want to use and then enters a value 1 and value 2 after which the arithmetic is performed and the answer is presented to the user onscreen.

## ../../statemac.pngFinite-State Machine



## Data Flow Diagram

# Requirements

The proposed second version of the software involved, most importantly, extending the simple calculator to include a Graphical User Interface (GUI). Other extensions included adding the capability of handling multiple operator precedence, to perform multiple calculations within one sum, and implementing a memory function to store previously entered calculations which would provide the user with a way of viewing at least the last input and output.

|  |  |
| --- | --- |
| V2 | MoSCoW Requirements and Risk Analysis |
| Must have | A Graphical User interface (GUI), meet necessary requirements for brief. |
| Should have | The ability to hand multiple operator precedence, and a memory function to store and retrieve the last calculation. More efficient code, thorough testing, |
| Could have | A user friendly design, additional functionality. |
| Would have | Efficient code, advanced ease of use, intuitive design. |

With the transition of command line based code to GUI centred software there are many adaptations that must be made to the code to facilitate this merger. Version 2 will improve upon version 1 of the software in that it will increase usability through implementation of a GUI. This will help those less computer literate to navigate the software. Version 2 will also be superior in that in collates the incremental changes made in the various iterations of version 1 and presents them in a more appealing and easy to use manner. The code will be more effective, efficient, and streamlined as opposed to the clunky natured, trial-based version 1.

# The Process

Version 1 of the software was classified under various incremental stages. V1.0 notes the initial calculator that could perform simple calculations. V1.1 describes the second change, in which the calculator was extended to incorporate unary operations and the increase in efficiency and usability. V2 depicts the transition to GUI in which code was drastically altered to facilitate the new design.

## Version 1.0

The simple C++ Calculator was created by using a basic switch/case statement. The first step was to include iostream and use the standard namespace. The main function was created, an integer with a return value of zero. A character variable was created for the operator and this was named ‘op’. Then two float variables were created for the two numbers inputs, these were called ‘a’ and ‘b’. Then a welcome line is printed followed by a request for the operator, which is stored in the variable ‘op’, followed by a request for the two numbers which were stored in ‘a’ and ‘b’ variables. A switch statement was created one case for each command or operator with each operator having the relevant operator symbol applied the character variables in each case.

## Version 1.1

This version was redesigned for several reasons. Firstly, to eliminate the need for multiple types of variable, this served to keep the code neat. Secondly and more importantly it allowed the calculator to be constructed using a loop so that the user had control over when the program terminates, whereas before the calculator would terminate due to the switch statement. This loop allows the user to perform as many calculations as they wish before exiting the calculator program. This program also included both the iostream and cmath libraries which provide a few more arithmetic functions than the previous version. This program also used the same namespace as the previous version.

The code for version 1.1 is very different. Global variables were created so that all blocks of code could reference them. For this iteration of the program all variables are integers and four were used. One for the operator chosen by the user, this was named ‘op’. Two for the numbers chosen by the user, named ‘value1’ and ‘value2’. Plus, a final variable for the result of the calculation, this was named ‘ans’.

The main function was then created and the first action was to print a line identifying authors. After this the loop was created, this was done using a while loop. The first part of the while loop printed text on the screen to welcome the user and then presented them with a numbered list of options allowing them to perform simple and unary calculations. The list also presents the option to exit the program. The input provided by the user is read and stored in the ‘op’ variable. The next part of the code is an if statement that allows the exit option to function. This is followed by an if statement to request for and read the numbers provided by the user. The if statement was used because two of the calculations only requires one number and the others require two numbers. The proceeding code contains an if statement that checks the value of the variable ‘op’ and performs an action based on what it is. If it is equal to the corresponding number, then the apt arithmetic is performed and the result stored in the variable ‘ans’. Finally, the answer is presented to the user and the loop is executed again. If the user chooses to exit, then they are presented with a farewell message and the program terminates.

## Version 2

The first step was for a decision to be made regarding what software package will be used to create the GUI for the calculator. For all intents and purposes two options existed, Visual Studio Code and QT. For this project QT was the application of choice.

A project was created and a GUI was constructed using buttons for each of the operators, making 8 total buttons. There were also three labels and textboxes, two of the textboxes for numbers A and B, with respective labels. And the final textbox being for the result of the calculation with a similar label. This interface design was copied from a hand drawn sketch.

Following this a click event was created for each of the buttons and an individual function was written for each function specifying the number of variables to be created, the values to be stored in those variables and then the action to be performed in order to calculate the result and store it in the relevant variable. Then the result was displayed to the user in the textbox for the result. After each operator had been assigned a function a clear button was given a function to reset the value of each of the textboxes to 0.

# The Results

Ultimately, the group were happy with the achieved results, however due to time constraints did not have the required time to effectively develop and integrate the remaining extensions requested. The final version was tested to ensure quality throughout and that the product stood up to its requirements. Given more time the calculator would be upgraded significantly to include the memory function and more arithmetic functions. And were it to be made a Version 3 would include a button for each number and be laid out like a more traditional analogue calculator.

|  |  |  |  |
| --- | --- | --- | --- |
| Test number | Test Description | Expected outcome | Actual outcome |
| 1 | Test that all numbers from one to nine are accepted inputs. | Calculations will be performed accurately. | Inputs are accepted and displayed as expected.   * Passed. |
| 2 | Test against other keyboard characters, a-z. | The calculator will not function with the letters entered. | Doesn’t perform a calculation with keyboard characters A-Z.   * Passed. |
| 3 | Test all other ASCII characters. | The characters will not be accepted as inputs by the calculator. | ASCII characters not accepted by the calculator   * Passed. |
| 4 | Test to see if symbolic letters and numbers from other languages work. | The calculator should refuse character input but accept numerical input regardless of language. | The calculator refuses character input and accepts numerical input regardless of language.   * Process. |

# ../../CT4017-Calc/Screenshot/C++%20Code%20btn7-8.png../../CT4017-Calc/Screenshot/C++%20Code%20btn4-6.pngCode and Documentation

