**COMP 5541**

**Tools and Techniques for Software Engineering**

**Winter 2016**

Deliverable 1

(Title) Eternity

Team F

(Official Names of Team Members Only)

Contents

[1. Details of Deliverable #1 4](#_Toc443611980)

[2. Sections 5](#_Toc443611981)

[2.1 Domain 5](#_Toc443611982)

[2.2 Interviews 5](#_Toc443611983)

[2.3 Personas 5](#_Toc443611984)

[2.4 Use Cases 5](#_Toc443611985)

[2.5 Implementation 10](#_Toc443611986)

[2.6 Testing 10](#_Toc443611987)

[3. Team Member Responsibilities 11](#_Toc443611988)

[4. References 12](#_Toc443611989)

[5. Appendix 13](#_Toc443611990)

[6. Glossary 14](#_Toc443611991)

# 1. Details of Deliverable #1

Calculators have been a fundamental tool in the advancement of technology since their invention. They provide a means for users to solve unfeasibly long arithmetic problems in a matter of seconds. In today’s world, many different types of calculators exist, each with their own set of purposes and functionalities.

The purpose of this deliverable will be to design and build a functioning scientific calculator application following a rigorous design methodology. A calculator is considered scientific if it can perform functions and operations which are commonly used in the scientific and engineering fields. These operations include, but are not limited to, trigonometric functions, logarithmic functions and exponential functions.

Along with the standard operators, the application should implement a set of five transcendental functions which have been hard coded by the team directly. Table 1 summarizes the sections included in this deliverable:

Table 1: Description of report contents

|  |  |
| --- | --- |
| Section | Description |
| Domain | The domain of the project, including which transcendental functions were implemented and the decision-making behind choosing one algorithm over another. |
| Interviews | The list of questions asked and a brief summary on the design of the interview process |
| Personas | Requirements for personas will be elicited |
| Use Cases | Description of the various use cases and how they were implemented |
| Implementation | Description on the general implementation of the application |
| Testing | The testing procedure and process will be explained |
| Glossary | A glossary including terms that may be unfamiliar to some |
| Meeting Details | Details of each meeting, including minutes |
| Appendix | Where various results will be included |

# 2. Sections

## 2.1 Domain

Sin

Implementation 1 (used): Taylor series expansion

Implementation 2:

Implementation 3:

PI

Implementation 1 (used):  [Madhava–Leibniz series](https://en.wikipedia.org/wiki/Leibniz_formula_for_pi). 21 terms to compute an approximation of π correct to 11 decimal places as 3.14159265359

Implementation 2:  [Ramanujan–Sato series](https://en.wikipedia.org/wiki/Ramanujan%E2%80%93Sato_series)

Implementation 3: Chudnovsky brothers using a variation of Ramanujan’s approximation

## 2.2 Interviews

## 2.3 Personas

## 2.4 Use Cases

The users can play a central role in the engineering of an interactive software system. Therefore, it is critical to precisely understand, identify and document the services that an interactive software system will provide from the viewpoint of its potential users [Kamthan, 2016]. One way to present an abstraction of user actions is through use cases, where possible user interactions with the system are elicited and analyzed.

In this section, the possible use cases will be enumerated in tabular form, providing a short description of the use case, information on any preconditions (if any) necessary, error situations, system state in the event of an error, and if the use case includes or extends any other use cases. It is important to note that the system does not need to differentiate between different types of users, and so specifications on the actors for each use case is not necessary. For the remainder of this document, current expression refers to the expression currently being built by the user and is where the user enters an expression, while previous expression refers to the expression built previously, which should be displayed above the current one.

Table 2: Build Expression Use Case

|  |  |
| --- | --- |
| Name: | Build Expression |
| Description: | The user enters an expression to calculate |
| Standard Process: | The user builds his expression using operators, operands and custom functions |
| Potential errors, from system point of view: | NA |
| <<Includes>> | NA |
| <<Extends>> | Input Left Parenthesis  Input Right Parenthesis  Input Operator  Input Number  Input Decimal Point  Delete Character  Delete Custom Function  Input Negation Operator  Input Custom Function |

Table 3: Compute Expression Use Case

|  |  |
| --- | --- |
| Name: | Compute Expression |
| Description: | The user presses “=” to compute what was in the current expression. This expression String will be displayed in the previous expression, while the answer will be displayed in the current expression. |
| Standard Process: | Compute the value of the current expression |
| Potential errors, from system point of view: | - If current expression begins with an operator, append it to previous expression and compute.  -If current expression does not begin with operator, overwrite previous expression and execute.  - If current expression finishes with an operator, append a 0 to it.  -If current expression is empty, do nothing.  -If # of open parenthesis > # of closed parenthesis, append appropriate number of closed parenthesis.  - If current expression has a math error (divide by 0 for example), display error message. |
| <<Includes>> | NA |
| <<Extends>> | NA |

Table 4: Input Left Parenthesis Use Case

|  |  |
| --- | --- |
| Name: | Input Left Parenthesis |
| Description: | The user wants to add an open parenthesis |
| Standard Process: | The user presses “(“ and appends it to the current expression |
| Potential errors, from system point of view: | Will not append if the user tries to add a “(“ anywhere except for an operator, another “(“ or at the start of a custom function. |
| <<Includes>> | NA |
| <<Extends>> | NA |

Table 5: Input Right Parenthesis Use Case

|  |  |
| --- | --- |
| Name: | Input Right Parenthesis |
| Description: | The user wants to add a closed parenthesis |
| Standard Process: | The user presses “)“ and appends it to the current expression |
| Potential errors: | Will not append unless it is following either a digit or another closed parenthesis |
| <<Includes>> | NA |
| <<Extends>> | NA |

Table 6: Input Decimal Point Use Case

|  |  |
| --- | --- |
| Name: | Input Decimal Point |
| Description: | The user wants to add a decimal point |
| Standard Process: | The user presses “.“ and appends it to the current expression |
| Potential errors: | Will not append unless it is after a digit |
| <<Includes>> | NA |
| <<Extends>> | NA |

Table 7: Backspace Use Case

|  |  |
| --- | --- |
| Name: | Execute a backspace |
| Description: | The user wants to delete the last character in the current expression |
| Standard Process: | The user presses the backspace button and the last character (digit, operand, function, parenthesis or decimal point) gets deleted |
| Potential errors: | NA |
| <<Includes>> | NA |
| <<Extends>> | NA |

Table 8: Input Operator Use Case

|  |  |
| --- | --- |
| Name: | Input Operator |
| Description: | The user wants to add either a +, -, x or ÷ |
| Standard Process: | - The user presses one of the operators and appends it to the current expression  - If there is a previous expression displayed and the user begins the current expression with an operator, the system will append the operator and anything after it to the previous expression |
| Potential errors: | - Negative operator will not append unless it is after a digit or any parenthesis  - The remaining operators will not append unless it is after a digit or closed parenthesis |
| <<Includes>> | NA |
| <<Extends>> | NA |

Table 9: Input Digit Use Case

|  |  |
| --- | --- |
| Name: | Input Digit |
| Description: | The user wants to add a digit |
| Standard Process: | The user presses a number from 0-9 and appends it to the current expression |
| Potential errors: | Will not work unless it is inserted after an open parenthesis, decimal point or another digit |
| <<Includes>> | NA |
| <<Extends>> | NA |

Table 10: Clear Operation Use Case

|  |  |
| --- | --- |
| Name: | Clear Operation |
| Description: | The user wants to clear both the current and previous expressions |
| Standard Process: | The user presses the “C” button and clears both the current and previous expressions |
| Potential errors: | NA |
| <<Includes>> | NA |
| <<Extends>> | NA |

Table 11: Clear Expression Operation Use Case

|  |  |
| --- | --- |
| Name: | Clear Expression Operation |
| Description: | The user wants to clear only the current expression |
| Standard Process: | The user presses the “CE” button and clears the current expression |
| Potential errors: | NA |
| <<Includes>> | NA |
| <<Extends>> | NA |

Table 12: Input Sine Function Use Case

|  |  |
| --- | --- |
| Name: | Input Sine Function |
| Description: | The user wants to add the sine function |
| Standard Process: | The user presses the “sin” button and appends a “sin(“ to the current expression |
| Potential errors: | Will not append unless it is after |
| <<Includes>> | NA |
| <<Extends>> | NA |

Table 13: Input ℯˣ Function Use Case

|  |  |
| --- | --- |
| Name: | Input ℯˣ Function |
| Description: | The user wants to add the ℯˣ function |
| Standard Process: | The user presses the “ℯˣ” button and appends a “ℯˣ “to the current expression |
| Potential errors: | Will not append unless it is after |
| <<Includes>> | NA |
| <<Extends>> | NA |

Table 14: Input √(x) Use Case

|  |  |
| --- | --- |
| Name: | Input √(x) Function |
| Description: | The user wants to add the √(x) function |
| Standard Process: | The user presses the “√(x)” button and appends a square root operator to the current expression |
| Potential errors: | Will not append unless it is after |
| <<Includes>> | NA |
| <<Extends>> | NA |

Table 15: Input 10ˣ Function Use Case

|  |  |
| --- | --- |
| Name: | Input 10ˣ Function |
| Description: | The user wants to add the 10ˣ function |
| Standard Process: | The user presses the “10ˣ” button and appends a “10ˣ” to the current expression |
| Potential errors: | Will not append unless it is after |
| <<Includes>> | NA |
| <<Extends>> | NA |

In order to restrict the user from entering syntactically wrong inputs inside the calculator, certain conditions needed to be specified in the source code. When the user tries to press a button, hence append a character, a regular expression is used as a pattern to match with the current expression and determine whether or not the button press is valid. These constraints are listed above in the Potential errors sections of the use case and are implemented in Android Studio once a button calls its OnClick() method. As for dealing with mathematically incorrect inputs, the exp4j object handles these errors and displays the appropriate error message.

## 2.5 Implementation

## 2.6 Testing

# 3. Team Member Responsibilities

# 4. References

[Kamthan 2016] COMP 5541 class notes, 2016.

# 5. Appendix

# 6. Glossary

**Abstraction**: A technique for managing complexity of computer systems by establishing a level of complexity on which a person interacts with the system, suppressing the more complex details below the current level

**Actor:** A role played by an entity external to the system, as perceived from the system’s point of view, when it uses the system

**Android:** a [mobile operating system](https://en.wikipedia.org/wiki/Mobile_operating_system) (OS) currently developed by [Google](https://en.wikipedia.org/wiki/Google) and designed primarily for [touchscreen](https://en.wikipedia.org/wiki/Touchscreen) mobile devices such as [smartphones](https://en.wikipedia.org/wiki/Smartphone) and [tablets](https://en.wikipedia.org/wiki/Tablet_computer).

**Android Studio:**  the official [integrated development environment](https://en.wikipedia.org/wiki/Integrated_development_environment) (IDE) for developing Android applications

**Append:** To add something as an attachment

**Domain:** A field of study that defines a set of common requirements, terminology, and functionality for any software.

**Git:** A widely used distributed revision control system for software development.

**GitHub:** A web-based Git repository hosting service.

**Implementation:** The process of putting a design or decision into effect.

**Java**: An Object-Oriented Programming language designed to have as few implementation dependencies as possible and developed by Suns Microsystems.

**Object-Oriented Programming:** A programming paradigm based on the concepts of data structures called objects, which store data known as attributes, and perform procedures known as methods.

**Persona:** A fictional but realistic user of a software system.

**Use Case:** A sequence of actions performed by a system, which yields an observable result of value to an actor of the system.

**User-Interface (UI)**: Everything designed for the purpose of a user to interact with the program, including display screen, messages, buttons, etc.

**Regular Expression:** A sequence of [characters](https://en.wikipedia.org/wiki/Character_(computing)) that define a search pattern, mainly for use in [pattern matching](https://en.wikipedia.org/wiki/Pattern_matching) with [strings](https://en.wikipedia.org/wiki/String_(computer_science)), or [string matching](https://en.wikipedia.org/wiki/String_matching).

**Transcendental function:** A function that does not satisfy any single-variable polynomial equation whose coefficients are themselves roots of polynomials.

**Wiki:** A piece of server software that allows users to freely create and edit Web page content using any Web browser.