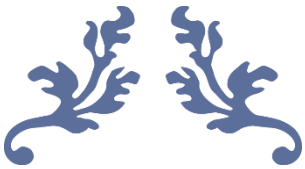
UNIVERSITÀ DEGLI STUDI
DI SALERNO



PROGRAMMABLE CALCULATOR

Requirements Engineering

29th NOVEMBER 2023

DE LUCA A. - D'AGOSTINO M. - DE LUCA D. - FESTA R.

- GROUP 21 -

Index

1 About

1.1 Team Information

Name	ID
D'Agostino Marco	*0612705817*
De Luca Aniello *(group leader)*	*0612705805*
De Luca Daniele	*0612705654*
Festa Raffaele	*0612705535*

1.2 Version

Revision	Date	Summary of changes
1.0	25/11/2023	First release

2 Introduction

2.1 Purpose

The purpose of this project is to make a calculator app that uses a Stack to work with both complex and real numbers. By manipulating the Stack, the app enables users to carry out a range of mathematical operations, providing a flexible tool for numerical computation. Moreover, the app's interface shows the top elements of the Stack to simplify the input of complex expressions. Variable storage allows users to store and retrieve values when making calculations.

2.2 Definition and Acronyms

Acronyms	Definition
IF	Individual functionality
BF	Business flow
DF	Data and data format
UI	User interface
IS	Interface to systems
FC	Further constraints
var	Letter Variables
op.	Operations with any values

1-AcronymsTable

3 System Features and Requirements Analysis

3.1 Requirements Analysis

BF-1: Enter Value

UI-1.1: For each component of the *value (op., numbers, var)* the user will see the *button* on the GUI.

UI-1.2: The entered values are displayed in the GUI *text-area*.

IS-1.1: Values *(op., numbers, var)* are entered by clicking on the associated *button*.

IF-1.1: Input of a complex number.

DF-1.1: Must be a complex number in Cartesian Form (es. 53+9j).

DF-1.2: May include real numbers written without imaginary part (es. 53, seen as 53+0j).

IF-1.2: Input of each mathematical *op*. is done by entering the name of the associated *op.*.

DF-1.3: Math *op.* will be identified by the following names:

Name	Operation Description
+	Sum the last two elements
-	Sub the last two elements
*	Multiply the last two elements
/	Divide the last two elements
sqrt	Does Square Root of the last element
±	Does Inversion Sign of the last element

2-OperationNamesView

IF-1.3: Input of each *op.* with *var* is done by entering the name of the *op.* followed by the *var* selected.

DF-1.4: The *User* can use 26 letters as *var (from A to Z)*.

DF-1.5: *Op.* on or with *var* happen by format: *op.name-var* (es. *+var, -var, *var, /var...)*.

DF-1.6: Other formats of *op.* strictly related to *var* are: *>var e <var*.

BF-2: Execution of the entered value

UI-2.1: Values are executed by clicking on the appropriate *button* on the GUI.

IF-2.1: The syntactically correct value can be executed.

UI-2.2: Changes are displayed in a dedicated area of the GUI.

IS-2.1: *Numerical Value* Execution→ the number is inserted at the top of the stack.

IS-2.2: *Op.* Execution→ the operation is executed, the operand and/or operands are removed from the stack, and the result is placed in the top position of the stack.

IS-2.3: *Op.Var* Execution → the matching *op.* is executed:

Operation	Description
>var	Takes the *top element* from the stack and saves it in the *var*
<var	Takes the element from the *var* and saves it in the *top of the stack*
+var	Adds to *var* the top element of the stack
-var	Subs to *var* the top element of the stack
**var*	Multiplies to *var* the top element of the stack
/var	Divides to *var* the top element of the stack
sqrt *var*	Does square root of the *var*
±*var*	Does sign inversion of the *var*

3-OperationView

BF-3: Perform Stack Manipulation

UI-3.1: Each function is displayed as a *button* on the GUI.
IS-3.1: Each function is executed by clicking the corresponding *button*.
IF-3.1: Option to remove all elements from stack with the "**clear**" function.
IF-3.2: Option to remove the last element of stack *(top e.)* with the "**drop**" function.
IF-3.3: Option to add the copy of the last element *(top e.)* to the stack with the "**dup**" function.
IF-3.4: Option to exchange the positions of the last two elements of the stack with the "**swap**" function.
IF-3.5: Option to add the copy of the *second last element* to the stack with the "***over***" function.

3.2 Non-functional Requirements (FC)

Performance requiremets:

1. System must be stack based.
2. Stack must be able to contain at least 12 elements.

Safety requirements:

3. The system must be usable and functional every time it is started up.

Usability requirements:

4. The interface must be intuitive and clear in its functions.
5. System must support RPN-type syntax.

Maintainability requirements:

6. Source Code must be well structured.
7. The project must be documented and commented.
8. It must be easy to fix bugs in the code.

Compatibility requirements:

9. The system must be able to run on any system where java is supported.

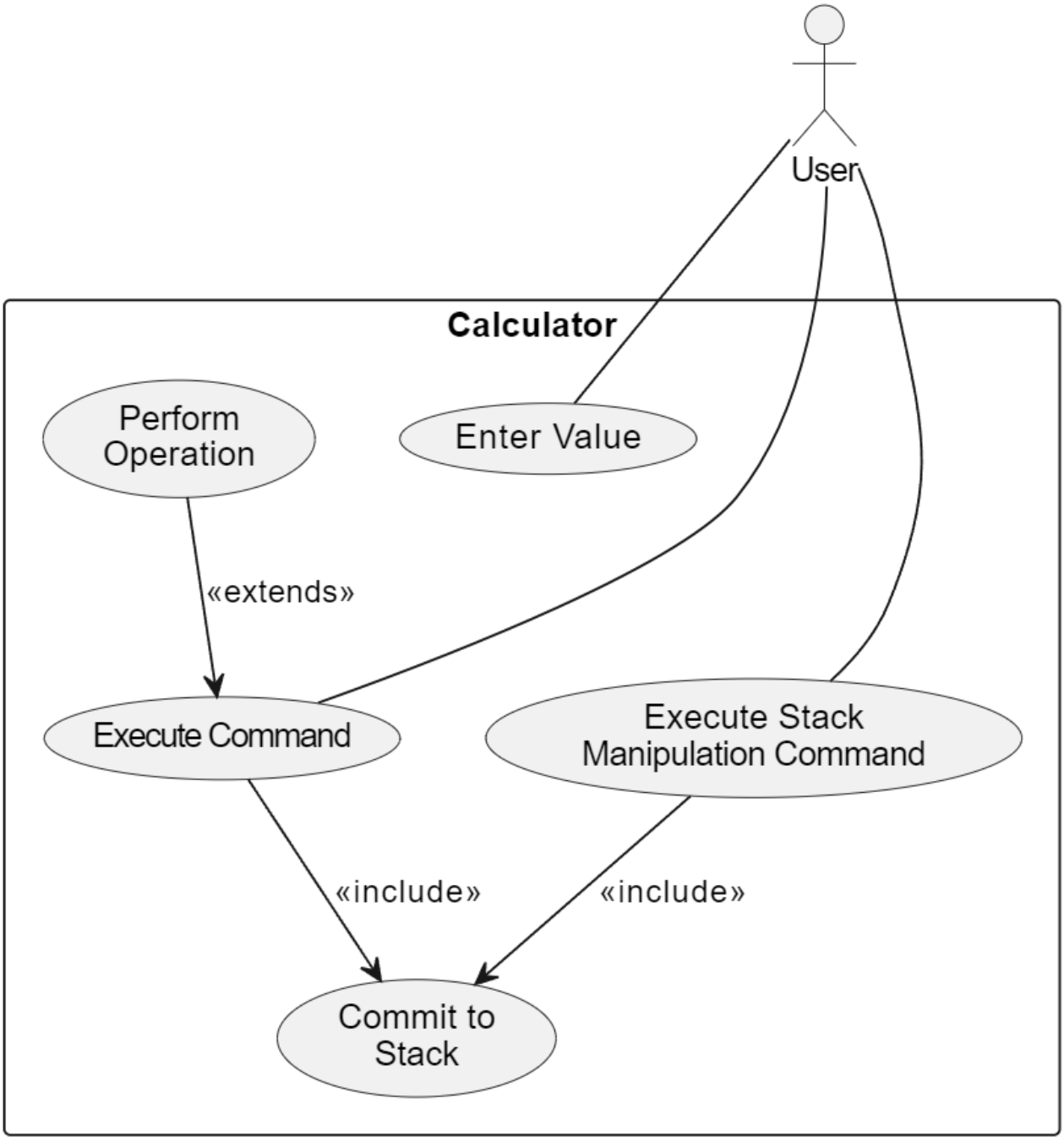
3.3 Requirements Prioritization List

Req-ID	Description	Priority
UI-1.1	For each component of the *value (op., numbers, var)* the user will see the *button* on the GUI	HIGH
UI-1.2	The entered values are displayed in the GUI text area	HIGH
IS-1.1	The input of the values is done by clicking on the respecting button	HIGH
IF-1.1	The user can input a complex number	MEDIUM
DF-1.1	The inputted complex number must be in a Cartesian Form	HIGH
DF-1.2	The inputted complex number can be specialized in natural number by omitting the imaginary part	LOW
IF-1.2	The user can input an operation name	HIGH
DF-1.3	The name *op.* inputted can be in symbols	MEDIUM
IF-1.3	The user can input a variable	HIGH
DF-1.4	The variable inputted can be a letter from A to Z	MEDIUM
DF-1.5	The user can input an operation with variables	HIGH
DF-1.5.1	The user can input specific operation that regards variables	MEDIUM
UI-2.1	The user can excute a value by clicking on a dedicated button	LOW
IF-2.1	The user can excute a value if it is syntactically correct	HIGH
UI-2.2	The stack changes are displayed in a part of the GUI	LOW
IS-2.1	The executed number value is pushed into the stack	HIGH
IS-2.2	The executed operation is done and the operands are deleted from the stack. The result is pushed in to the stack	HIGH
IS-2.3	The executed variable-operation is done and eventually changes to the stack are saved	HIGH
UI-3.1	Every stack manipulation function is visualized by a dedicated button on the GUI	HIGH
IS-3.1	Every stack manipulation function can be done by pressing the dedicated button	HIGH
IF-3.1	The user can clear the stack by pressing the button of the **clear** function	MEDIUM
IF-3.2	The user can remove the last element of the stack by pressing button of the **drop** function	MEDIUM
IF-3.3	The user can add a copy of the last element of the stack at the top by pressing the button of the **dup** function	MEDIUM
IF-3.4	The user can exchange the position of the last two elements of the stack by pressing the button of the **swap** function	MEDIUM
IF-3.5	The user can add a copy of the second last element to the stack by pressing the button of the **over** function	MEDIUM

4-ReqPriorityList

4 Requirements Definition

4.1 Use Case Diagram



1-Use Case Diagram (made with PlantUML)

4.2 Use Case Definition

UC-1	Enter Value	
Description	The system allows users to enter values through a graphical user interface (GUI). Each component of the value entered, such as operations, numbers, and variables, is represented by a corresponding button on the GUI. The values entered are displayed in the text area of the GUI. Users can enter complex numbers, operation names, and operations involving variables.	
Precondition	The GUI is online ad ready to get inputs.	
Ordinary Sequence	**Step**	**Action**
	1	The user interacts with the GUI to enter a value.
Postcondition	The entered value is displayed in the GUI text-area.	
Exceptions	No Exception	

UC-2	Execute Command	
Description	The System allows the elaboration of the entered values *(op., numbers, var command)* through the click on the respective button on the GUI.	
Precondition	The GUI is active, ready to get an input and the user entered the values in the text-area.	
Ordinary Sequence	**Step**	**Action**
	1	The user executes the entered value with a click on the dedigned GUI button.
	2	The System processes and executes (performs) the value.
Postcondition	The value is processed and executed	
Exceptions	**Step**	**Action**
	1	Invalid or incomplete value in text-area. The System warns the user and requires a correct input.

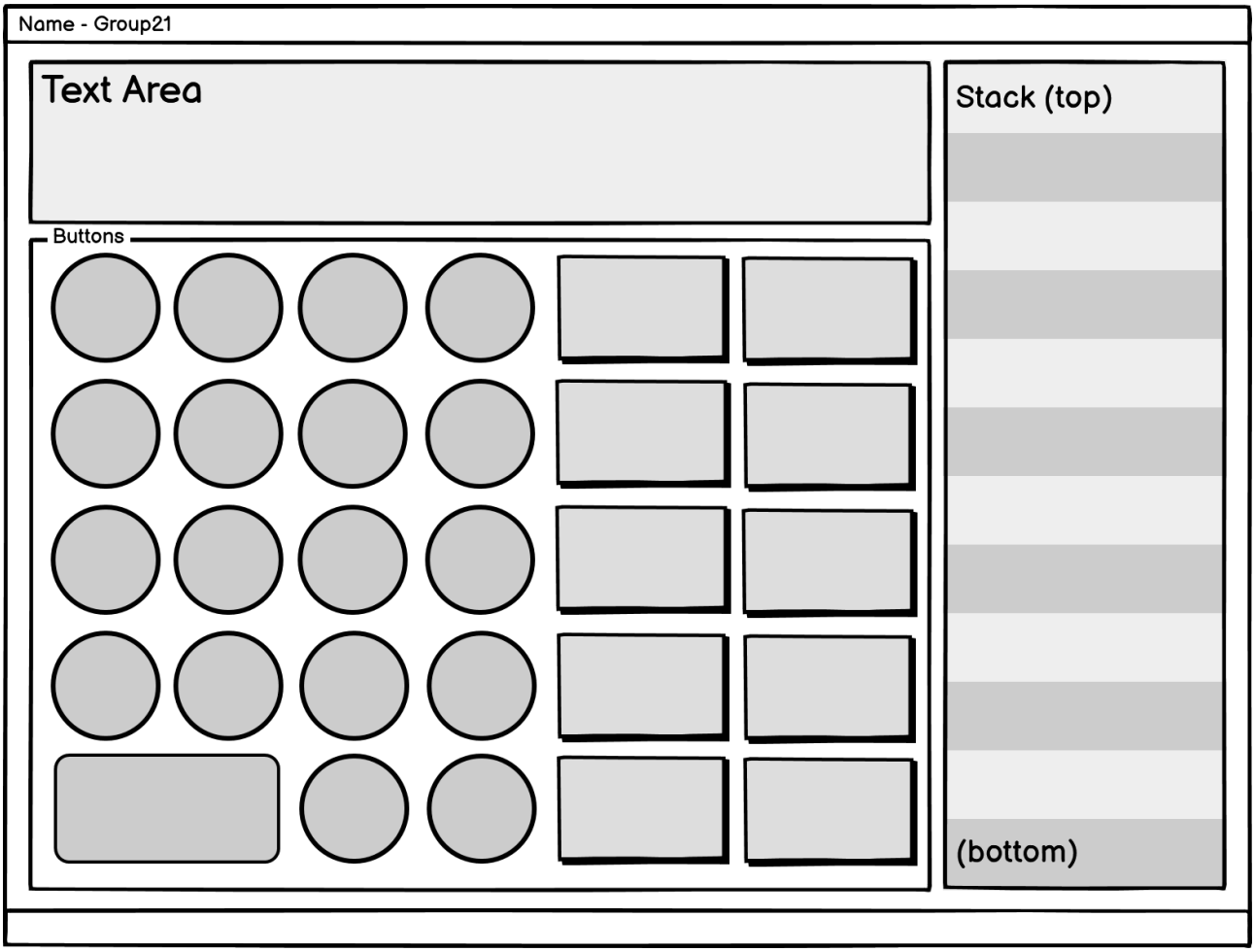
UC-3	Execute Stack Manipulation	
Description	The System allows stack manipulation through a *Grafic User Interface* (GUI). Each manipulation function is represented on the GUI by a button. Functions are executed clicking on respective button.	
Precondition	The GUI is active and ready to get an input.	
Ordinary Sequence	**Step**	**Action**
	1	The user interacts with the GUI to execute manipulation *command* on the stack.
	2	The system processes and performs the chosen operation.
Postcondition	The chosen stack manipulation *command* has been performed.	
Exceptions	**Step**	**Action**
	1	There aren't enough elements in the stack to perform the manipulation *command*.

UC-4	Commit to Stack	
Description	The system applies each execution of the value to the stack and any manipulation of it.	
Precondition	The GUI is active and the stack is either empty or contains values. Operations or values have been executed.	
Ordinary Sequence	**Step**	**Action**
	1	The system identifies the type of execution.
	2	The system applies the esecution changes to the stack.
Postcondition	The resulting changes are applied to the stack and displayed in the dedicated GUI area.	
Exceptions	**Step**	**Action**
	2	The stack's memory is full.

UC-5	Perform Operation	
Description	The system execute the operation recognized in the **UC-2**.	
Precondition	The system executed a mathematic operator or a variable operation.	
Ordinary Sequence	**Step**	**Action**
	1	The system recognize the type of operation.
	2	The system execute the operation taking the necessary operand from the stack.
Postcondition	The system displays the result of the operation	
Exceptions	**Step**	**Action**
	1	The execute is not valid in case of indeterminate form.
	2	There aren’t enought operands in the stack.

5 Interface Mock-Up

The interface is composed of a display organized in this way:



2-InterfaceMock-Up

6 Tracebility Matrix

Tracebility Matrix – Group 21 – v1.0				
Req-ID	**Design**	**Code**	**Test**	**Related Requirements**
UI-1.1				N/A
UI-1.2				N/A
IS-1.1				N/A
IF-1.1				N/A
DF-1.1				**IF-1.1**
DF-1.2				**IF-1.1**
IF-1.2				N/A
DF-1.3				**IF-1.2**
IF-1.3				N/A
DF-1.4				**IF-1.3**
DF-1.5				**IF-1.3**
DF-1.6				**IF-1.3**
UI-2.1				N/A
IF-2.1				N/A
UI-2.2				**IF-2.1**
IS-2.1				**IF-2.1**
IS-2.2				**IF-2.1**
IS-2.3				**IF-2.1**
UI-3.1				N/A
IS-3.1				N/A
IF-3.1				N/A
IF-3.2				N/A
IF-3.3				N/A
IF-3.4				N/A
IF-3.5				N/A