Test Plan for the Companion Cube Calculator (C^3) Tool

Geneva Smith

December 19, 2017

1 Revision History

Date	Version	Notes
December 19, 2017	2.0	Updated plan to reflect current implementation
November 13, 2017	1.0.1	Addition of missing test cases
October 25, 2017	1.0	Initial draft completed

2 Symbols, Abbreviations and Acronyms

Symbol	Description
C^3	Companion Cube Calculator
GUI	Graphical User Interface
IDE	Integrated Development Environment
SRS	Software Requirements Specification
T	Test

Contents

1 Revision History **i**

2 Symbols, Abbreviations and Acronyms **ii**

3 General Information **1**
- 3.1 Purpose . . . 1
- 3.2 Scope . . . 1
- 3.3 Overview of Document . . . 2

4 Plan **3**
- 4.1 Software Description . . . 3
- 4.2 Test Team . . . 3
- 4.3 Automated Testing Approach . . . 3
- 4.4 Verification Tools . . . 4
- 4.5 Non-Testing Based Verification . . . 4

5 System Test Description **6**
- 5.1 Tests for Functional Requirements . . . 6
  - 5.1.1 Getting User Inputs . . . 6
  - 5.1.2 Conditioning User Inputs . . . 12
  - 5.1.3 Correctness of Component Calculations . . . 21
  - 5.1.4 Composing Operators . . . 33
  - 5.1.5 Presentation of Calculations to the User . . . 36
- 5.2 Tests for Non-Functional Requirements . . . 38
  - 5.2.1 Usability . . . 38
- 5.3 Traceability Between Test Cases and Requirements . . . 39

6 Unit Testing **41**
- 6.1 Control Flow Unit Tests . . . 41
- 6.2 User Input Unit Tests . . . 43
- 6.3 Interval Data Structure and Interval Conversion Unit Tests . . . 46
- 6.4 Equation Data Structure and Equation Conversion Unit Tests . . . 49
- 6.5 Variable Consolidation Unit Tests . . . 55
- 6.6 Operator Data Structure and Range Solver Unit Tests . . . 57
- 6.7 Output Unit Tests . . . 64

7 Appendix: Usability Testing **67**
7.1 Tasks 67
7.2 Questionnaire 67
7.2.1 Test Results (Round values as necessary) 67
7.2.2 Software Usability 68

List of Tables

1 Traceability Matrix Showing the Connections Between Requirements and Test Suites 39

List of Figures

1 Traceability Graph Showing the Connections Requirements and Test Suites 40

3 General Information

This document is a software test plan for the Companion Cube Calculator (C^3), a mathematical tool which determines the range of a user-specified function given the domains of the function's variables. The calculations are performed using interval arithmetic.

3.1 Purpose

This document describes the software test plan for the C^3 tool and is informed by its SRS documentation (version 2.0) and resulting implementation, including a description of automated testing approach, tools, and test cases. The purpose of documenting this information is to guide the product testing for the initial product release and to provide a basis for regression testing as further developments are made to the C^3 tool.

This document is intended for readers who wish to test the initial version of the product, as well as those who want to refine and expand the tool. As changes are made to the C^3 tool, these test cases will form the foundation of regression testing and will help ensure that any changes made to the product do not affect its original purpose or core functionality.

This test plan is intended to be used for version 1.0 of the C^3 tool, and will only contain test information related to the product description in the product version's SRS documentation (version 2.0). Any functionality defined beyond the scope of the SRS document are not included in this version of the test plan.

All documentation and the associated release of the tool can be found at https://github.com/GenevaS/CAS741.

3.2 Scope

The C^3 tool relies on interval arithmetic in the domain of real numbers ($\mathbb{R}$), which means that many of its functions can be empirically tested. This plan contains testing for both specific and non-specific implementation aspects of the program. The implementation-specific tests are included to achieve 100% code coverage to ensure that all code paths are reachable and produce expected outcomes. Non-specific implementation testing focuses on the correctness of the program with respect to its original purpose and its adherence to the SRS document.

The purpose of this plan is to modularize testing of the C^3 tool so that it is easy to maintain and improve. It also forms the foundation of the kinds of be-

haviours that the tool should exhibit when it encounters malformed user inputs and unexpected values during its calculations.

3.3 Overview of Document

This document presents a general description of the C^3 tool to establish the context of the test plan and a list of team members responsible for executing it. This background information is followed by a high-level description of the test plan, including the automated testing approach, verification tools, and non-testing based verification methods that will be used. The general plan outline is followed by a description of the system test which is designed to determine if the requirements from the associated SRS document are satisfied. The final section describes implementation-specific tests that were implemented to achieve 100% code coverage.

4 Plan

This section describes the test plan for the C^3 tool from a high-level perspective, outlining the methodologies and tools that will be used during the verification process, and the team responsible for executing the plan.

4.1 Software Description

The C^3 tool is a stand-alone application for calculating the mathematical range of a user-defined equation given the domains of the equation's component variables. Equation ranges and variable domains are represented by intervals, where intervals are defined as a set of consecutive values bounded by a minimum and maximum value. The minimum and maximum values for any given interval must be defined. To perform its calculations, the C^3 tool uses interval arithmetic.

The purpose of this product is to produce accurate results when they can be found with respect to the host machine's floating-point precision. If a result cannot be found, the tool is expected to communicate the reason back to the user. Exact results are not required, as the tested equations are expected to the implemented in a software environment that has no impact on the external world. These tasks must be completed while presenting all communications to the user in standard function and interval notation.

4.2 Test Team

The test team assigned to implement this plan is Geneva Smith.

4.3 Automated Testing Approach

The C^3 tool will almost exclusively rely on automated test approaches for its verification process, with the notable exceptions of the non-functional requirements for correctness, verifiability, usability and maintainability described in the SRS document (version 2.0). The verification of correctness cannot be tested by a machine because it relies on mathematical proofs and the remaining non-functional requirements of verifiability, usability, and maintainability are intended for a human audience. These types of verifications are best handled by manual tests and user studies.

The functional requirements tests (Section 5.1) focus on the behaviour that is expected at each stage of the C^3 tool's work flow. Since these behaviours are inter-

nal to the tool and do not require human guidance, these behaviours can be tested automatically using a series of black box unit tests. Some of the non-functional requirements such as robustness (Section 5.2), can also be tested using automated approaches because they focus on empirically testable data, such as data constraints.

Many of the unit tests are designed to be used as part of integration testing, and the system will be progressively tested starting from the creation of data structures, to conversion processes, to input and output testing. Outputs include messages informing the user of erroneous behaviours or malformed inputs, and the calculated values and resulting equation tree if a no errors are detected.

The remaining unit tests are designed to test the correctness of the tool's calculation. Some of the tests check simple, two variable equations to ensure that the individual calculation types are behaving correctly. The remaining tests will ensure that equations with multiple operators are being decomposed correctly by comparing it to expected results. Once a set of equations has been selected for this task, regression testing becomes available to check further developments to the C^3 tool.

4.4 Verification Tools

The C# programming language has been chosen for the development of the C^3 tool because of its GUI building capabilities. The supporting IDE, Visual Studio 2017 (Enterprise Edition), comes with a number of built-in test tools, including:

- **A unit test framework and management system**
  This forms the bulk of the automated testing approach and will be used for unit, integration, system, and regression testing. The IDE can be configured so that existing unit tests are run automatically when the program is compiled.

- **Code coverage tools**
  The code coverage tools in Visual Studio is connected to the test management system and can help identify code that is not being run by any unit test.

4.5 Non-Testing Based Verification

There are a few non-functional requirements that cannot be tested automatically, but are still worth testing to verify their correctness:

- Correctness can be tested using mathematical proofs to verify the correctness of equation decomposition and order of operations.
- Verifiability, usability and maintainability can be verified using user studies tailored for each requirement. This plan specifically contains a user study description for the usability requirements (Appendix 7).

5 System Test Description

The system testing of the C^3 tool will focus on inspecting and conditioning user inputs, verifying that the correct values are being produced by program calculations, and ensuring that the C^3 tool is able to produce descriptive outputs for both valid and invalid inputs. These tests are divided into sections based on modules so that integration testing can arise naturally from the gradual addition of components.

5.1 Tests for Functional Requirements

The functional requirement tests are broken into four main groups. The "Getting User Inputs" tests (5.1.1) test to make sure that no syntactic errors exist in the user's inputs that can be reasonably caught by the tool. The "Conditioning User Inputs" tests convert the user's $D(v), v \in V$ into internal data structures and parses $f(V)$ into a semantically correct representation following BEDMAS rules. The "Composing Operators" tests help prove that the composition of individual operators will produce the correct solution $R(f(V))$. Finally, the "Presentation of Calculations to the User" tests ensure that the C^3 tool follows the rules for significant figures and precision.

Each test case is associated with specific modules so that developers and maintainers have an idea of where to being their debugging process should a test fail.

5.1.1 Getting User Inputs

This test suite is designed to determine if the R1 (Accepting $f(V)$ and $D(v), v \in V$ as direct input or read from a file) functional requirement and any associated data constraints from R3 are satisfied.

Data that is read from a file is converted into the same format as the inputs for direct input methods, so they are not considered in the functional requirements testing explicitly. Complete unit tests for file I/O related functionality can be found in the User Input unit tests (Section 6.2).

User Input Tests

1. test-directinput

   Type: Functional, Automatic, System, Regression

   Initial State: New Session

Input: "x+y", "x,2,4 \ny,3,5"

Output: "[5, 9]",
"+- {+}
| +- {VAR} x
| +- {VAR} y"

User Message: Range calculated successfully.

How test will be performed: Unit Test

Associated Module: Control Flow

2. test-input_functionAsConstant

   Type: Functional, Automatic, Integration

   Initial State: New Session

   Input: "42"

   Output: [42, 42]

   User Message: Warning: The user equation is a constant value and the range will only include this value.

   How test will be performed: Unit Test

   Associated Module: Equation Conversion

User Inputs (Missing, Incomplete, and Invalid Equation)

1. test-input_noFunction

   Type: Functional, Automatic, Integration

   Initial State: New Session

   Input: "", "x,2,4 \ny,3,5"

   Output: $successCode = -3$

   User Message: Error: Unrecognized sequence encountered during Atomic Equation parsing. Remaining equation =

   How test will be performed: Unit Test

   Associated Module: Control Flow

User Inputs (Missing, Incomplete, and Invalid Domains)

1. test-input_noDomain

   Type: Functional, Automatic, Integration

   Initial State: New Session

   Input: `"x+y"`, `"x,2,4"`

   Output: $successCode = -2$

   User Message: Error: Cannot find intervals for variable name(s): y.

   How test will be performed: Unit Test

   Associated Module: Variable Consolidation

2. test-input_nonNumberInDomain

   Type: Functional, Automatic, Integration

   Initial State: New Session

   Input: `"x+y"`, `"x,2,4 \ny,a,5"`

   Output: $\overset{\frown}{+}$, { }

   $x \quad y$

   User Message: Error: The string provided for the minimum bound cannot be converted to a real number.

   How test will be performed: Unit Test

   Associated Module: Interval Conversion

3. test-input_missingMinDomainValue

   Type: Functional, Automatic, Integration

   Initial State: New Session

   Input: `"x+y"`, `"x,,4 \ny,a,5"`

   Output: $\overset{\frown}{+}$, { x = [4,4], y = [3,5]}

   $x \quad y$

User Message: Warning: No minimum interval bound given. Setting it to the same value as the maximum bound.

How test will be performed: Unit Test

Associated Module: Interval Conversion

4. test-input_missingMaxDomainValue

   Type: Functional, Automatic, Integration

   Initial State: New Session

   Input: `"x+y"`, `"x,2,4 \ny,3,"`

   Output: $\overset{\frown}{+}$, { x = [2,4], y = [3,3]}

   $x \quad y$

   User Message: Warning: No maximum interval bound given. Setting it to the same value as the minimum bound.

   How test will be performed: Unit Test

   Associated Module: Interval Conversion

User Inputs (Extraneous Information)

1. test-input_variableNotInFunction

   Type: Functional, Automatic, Integration

   Initial State: New Session

   Input: `"x+y"`, `"x,2,4 \ny,3,5 \nz,6,7"`

   Output: $\overset{\frown}{+}$, { x = [2,3], y = [4,5] }

   $x \quad y$

   User Message: Warning: Extraneous variables found in interval list (z).

   How test will be performed: Unit Test

   Associated Module: Variable Consolidation

Variable Names

1. test-input_simpleVariableName
   Type: Functional, Automatic, Integration
   Initial State: New Session
   Input: `"x+y"`
   Output: { `"x", "y"` }
   User Message: -
   How test will be performed: Unit Test
   Associated Module: Equation Conversion

2. test-input_multiCharacterVariableName1
   Type: Functional, Automatic, Integration
   Initial State: New Session
   Input: `"x1+y"`
   Output: { `"x1", "y"` }
   User Message: -
   How test will be performed: Unit Test
   Associated Module: Equation Conversion

3. test-input_multiCharacterVariableName2
   Type: Functional, Automatic, Integration
   Initial State: New Session
   Input: `"x_+y"`
   Output: { `"x_", "y"` }
   User Message: -
   How test will be performed: Unit Test
   Associated Module: Equation Conversion

4. test-input_multiCharacterVariableName3
   Type: Functional, Automatic, Integration
   Initial State: New Session
   Input: `"x_1+y"`
   Output: { `"x_1", "y"` }
   User Message: -
   How test will be performed: Unit Test
   Associated Module: Equation Conversion

5. test-input_multiCharacterVariableName4
   Type: Functional, Automatic, Integration
   Initial State: New Session
   Input: `"x_i+y"`
   Output: { `"x_i", "y"` }
   User Message: -
   How test will be performed: Unit Test
   Associated Module: Equation Conversion

6. test-input_multiCharacterVariableName5
   Type: Functional, Automatic, Integration
   Initial State: New Session
   Input: `"x‘‘+y"`
   Output: { `"x‘‘", "y"` }
   User Message: -
   How test will be performed: Unit Test
   Associated Module: Equation Conversion

7. test-input_multiCharacterVariableName6

   Type: Functional, Automatic, Integration

   Initial State: New Session

   Input: `"xy+y"`

   Output: { `"xy", "y"` }

   User Message: -

   How test will be performed: Unit Test

   Associated Module: Equation Conversion

5.1.2 Conditioning User Inputs

This test suite is designed to determine if the R2 (Converting each $D(v)$ into DD1) and R4 (Decomposing $f(V)$ into two-operand equations following BEDMAS rules) functional requirements and the associated data constraints from R3 are satisfied.

Unexpected Domain Ordering

1. test-parse_domainOrder

   Type: Functional, Automatic, Integration

   Initial State: New Session

   Input: `"x,2,4 \ny,3,-5"`

   Output: $\{x = [2, 4], y = [-5, 3]\}$

   User Message: WARNING: Value provided for intervals are not in increasing order.The values have been exchanged to maintain the interval ordering.

   How test will be performed: Unit Testing

   Associated Module: Interval Data Structure

2. unittest-intervaldatastructuresethighmin

   Type: Functional, Automatic, Unit

   Initial State: New Session

Input: $x = [3, 4]$

Changes: $min = 5$

Output: $x = [4, 5]$

User Message: WARNING: Value provided for minimum bound is greater than the current maximum bound. The values have been exchanged to maintain the interval ordering.

How test will be performed: Unit Testing

Associated Module: IntervalStruct

3. unittest-intervaldatastructuresetlowmax

   Type: Functional, Automatic, Unit

   Initial State: New Session

   Input: $x = [3, 4]$

   Changes: $max = -1$

   Output: $x = [-1, 3]$

   User Message: WARNING: Value provided for maximum bound is smaller than the current minimum bound. The values have been exchanged to maintain the interval ordering.

   How test will be performed: Unit Testing

   Associated Module: IntervalStruct

Creating Equation Trees

1. test-parse_addition

   Type: Functional, Automatic, Integration

   Initial State: New Session

   Input: `"x+y"`

   Output: $+ :$, { `"x", "y"` }

   $VAR : x \quad VAR : y$

User Message: -

How test will be performed: Unit Testing

Associated Module: Equation Conversion

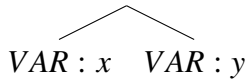
2. test-parse_subtraction

   Type: Functional, Automatic, Integration

   Initial State: New Session

   Input: `"x-y"`

   Output: $-$: , { `"x", "y"` }

   User Message: -

   How test will be performed: Unit Testing

   Associated Module: Equation Conversion

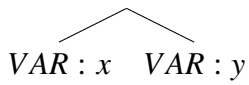
3. test-parse_multiplication

   Type: Functional, Automatic, Integration

   Initial State: New Session

   Input: `"x*y"`

   Output: $*$: , { `"x", "y"` }

   User Message: -

   How test will be performed: Unit Testing

   Associated Module: Equation Conversion

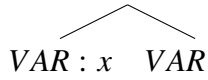
4. test-parse_division

   Type: Functional, Automatic, Integration

   Initial State: New Session

Input: `"x/y"`

Output: $/ :$, { `"x", "y"` }



User Message: -

How test will be performed: Unit Testing

Associated Module: Equation Conversion

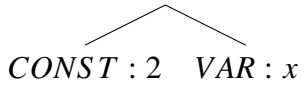
5. test-parse_intervalexponents

   Type: Functional, Automatic, Integration

   Initial State: New Session

   Input: `"2^x"`

   Output: $\wedge :$, { `"x"` }



   User Message: -

   How test will be performed: Unit Testing

   Associated Module: Equation Conversion

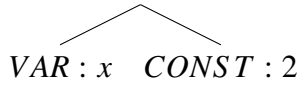
6. test-parse_intervalbase

   Type: Functional, Automatic, Integration

   Initial State: New Session

   Input: `"x^2"`

   Output: $\wedge :$, { `"x"` }



   User Message: -

   How test will be performed: Unit Testing

   Associated Module: Equation Conversion

Constant Values

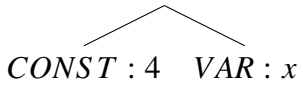
1. test-parse_constantValue1

   Type: Functional, Automatic, Integration

   Initial State: New Session

   Input: `"4+x"`

   Output: $+:$, { `"x"` }

   User Message: -

   How test will be performed: Unit Testing

   Associated Module: Equation Conversion

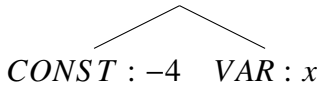
2. test-parse_constantValue2

   Type: Functional, Automatic, Integration

   Initial State: New Session

   Input: `"-4+x"`

   Output: $+:$, { `"x"` }

   User Message: -

   How test will be performed: Unit Testing

   Associated Module: Equation Conversion

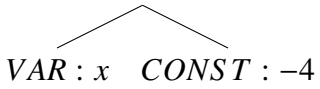
3. test-parse_constantValue3

   Type: Functional, Automatic, Integration

   Initial State: New Session

   Input: `"x/-4"`

Output: $/:$, { `"x"` }



User Message: -

How test will be performed: Unit Testing

Associated Module: Equation Conversion

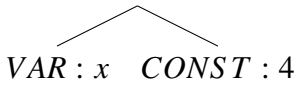
4. test-parse_constantValue4

   Type: Functional, Automatic, Integration

   Initial State: New Session

   Input: `"x+4"`

   Output: $+:$, { `"x"` }



   User Message: -

   How test will be performed: Unit Testing

   Associated Module: Equation Conversion

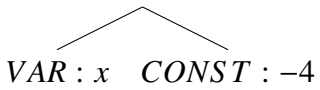
5. test-parse_constantValue5

   Type: Functional, Automatic, Integration

   Initial State: New Session

   Input: `"x+-4"`

   Output: $+:$, { `"x"` }



   User Message: -

   How test will be performed: Unit Testing

   Associated Module: Equation Conversion

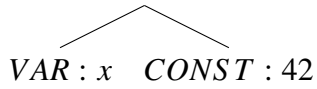
6. test-parse_constantValue6

   Type: Functional, Automatic, Integration

   Initial State: New Session

   Input: `"x+42"`

   Output: $+:$, { `"x"` }

   User Message: -

   How test will be performed: Unit Testing

   Associated Module: Equation Conversion

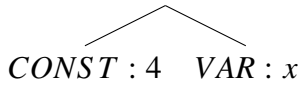
7. test-parse_implicitMultiplication

   Type: Functional, Automatic, Integration

   Initial State: New Session

   Input: `"4x"`

   Output: $*:$, { `"x"` }

   User Message: Warning: Encountered an implicit multiplication of a constant value and a variable. Expanding with explicit operator.

   How test will be performed: Unit Testing

   Associated Module: Equation Conversion

Precedence of Brackets

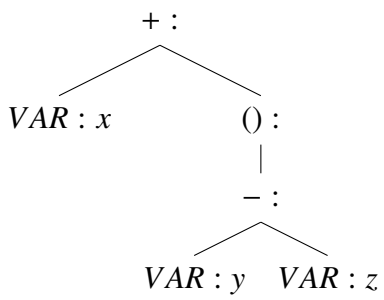
1. test-parse_brackets1

   Type: Functional, Automatic, Integration

   Initial State: New Session

   Input: `"x+(y-z)"`

Output:



User Message: -

How test will be performed: Unit Testing

Associated Module: Equation Conversion

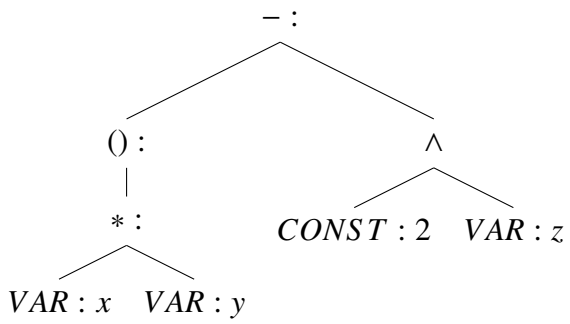
2. test-parse_brackets2

   Type: Functional, Automatic, Integration

   Initial State: New Session

   Input: `"(x*y)-2^z"`

   Output:



   User Message: -

   How test will be performed: Unit Testing

   Associated Module: Equation Conversion

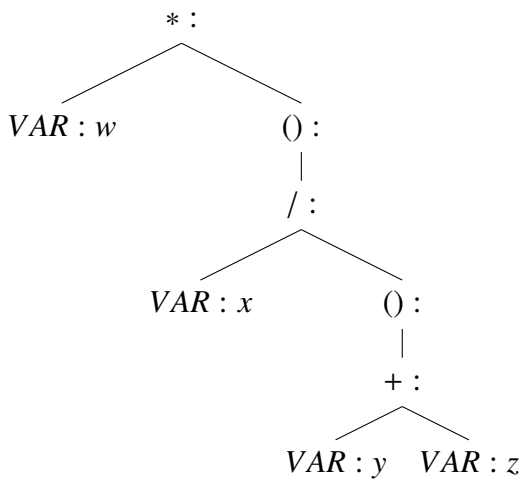
3. test-parse_brackets3

   Type: Functional, Automatic, Integration

   Initial State: New Session

   Input: `"w*(x/(y+z))"`

Output:



User Message: -

How test will be performed: Unit Testing

Associated Module: Equation Conversion

Open Brackets

1. test-parse_openLeftBracket

   Type: Functional, Automatic, Integration

   Initial State: New Session

   Input: `"x+y-z)"`

   Output: *NULL*

   User Message: Error: Unrecognized sequence encountered during Atomic Equation parsing. Remaining equation =)

   How test will be performed: Unit Testing

   Associated Module: Equation Conversion

2. test-parse_openRightBracket

   Type: Functional, Automatic, Integration

   Initial State: New Session

Input: `"x+(y-z"`

Output: $NULL$

User Message: Error: Could not find the end of the equation.

How test will be performed: Unit Testing

Associated Module: Equation Conversion

5.1.3 Correctness of Component Calculations

This test suite is designed to determine if the R5 (Solving each component equation in $f(V)$) and R7 (Ensuring that the calculated result is represented in closed, interval form) functional requirements, and the associated data constraints from R8 are satisfied.

Correctness of Addition, Subtraction, and Multiplication

1. test-calculate_addition

   Type: Functional, Automatic, Integration

   Initial State: New Session

   Input: $+:$, $\{x = [2,4], y = [3,5]\}$

   $VAR : x \quad VAR : y$

   Output: $[5,9]$

   User Message: -

   How test will be performed: Unit Testing

   Associated Module: Range Solver

2. test-calculate_additionconstant

   Type: Functional, Automatic, Integration

   Initial State: New Session

   Input: $+:$, $\{x = [2,4]\}$

   $VAR : x \quad CONST : 4$

Output: $[6, 8]$

User Message: -

How test will be performed: Unit Testing

Associated Module: Range Solver

3. test-calculate_subtraction

   Type: Functional, Automatic, Integration

   Initial State: New Session

   Input: $-$: , $\{x = [2, 4], y = [3, 5]\}$

   $VAR : x \quad VAR : y$

   Output: $[-1, -1]$

   User Message: -

   How test will be performed: Unit Testing

   Associated Module: Range Solver

4. test-calculate_multiplication1

   Type: Functional, Automatic, Integration

   Initial State: New Session

   Input: $*$: , $\{x = [2, 4], y = [3, 5]\}$

   $VAR : x \quad VAR : y$

   Output: $[6, 20]$

   User Message: -

   How test will be performed: Unit Testing

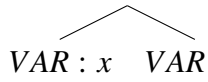
   Associated Module: Range Solver

5. test-calculate_multiplication2

   Type: Functional, Automatic, Integration

Initial State: New Session

Input: $*:$, $\{x = [-1, 3], y = [-3, 5]\}$



Output: $[-9, 15]$

User Message: -

How test will be performed: Unit Testing

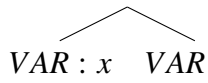
Associated Module: Range Solver

6. test-calculate_multiplication3

   Type: Functional, Automatic, Integration

   Initial State: New Session

   Input: $*:$, $\{x = [-3, -1], y = [-5, -2]\}$



   Output: $[2, 15]$

   User Message: -

   How test will be performed: Unit Testing

   Associated Module: Range Solver

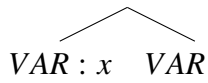
Correctness of Division

1. test-calculate_divisionPositiveDivisor1

   Type: Functional, Automatic, Integration

   Initial State: New Session

   Input: $/:$, $\{x = [2, 4], y = [3, 5]\}$



   Output: [0.4, 1.3333333333333333]

   User Message: -

How test will be performed: Unit Testing

Associated Module: Range Solver

2. test-calculate_divisionPositiveDivisor2

   Type: Functional, Automatic, Integration

   Initial State: New Session

   Input: $/$: , $\{x = [0, 4], y = [3, 5]\}$

   $VAR : x \quad VAR : y$

   Output: $[0, 1.3333333333333333]$

   User Message: -

   How test will be performed: Unit Testing

   Associated Module: Range Solver

3. test-calculate_divisionPositiveDivisor3

   Type: Functional, Automatic, Integration

   Initial State: New Session

   Input: $/$: , $\{x = [-1, 4], y = [3, 5]\}$

   $VAR : x \quad VAR : y$

   Output: $[-0.3333333333333333, 0.8]$

   User Message: -

   How test will be performed: Unit Testing

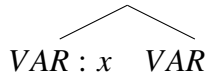
   Associated Module: Range Solver

4. test-calculate_divisionPositiveDivisor4

   Type: Functional, Automatic, Integration

   Initial State: New Session

Input: $/:$, $\{x = [-2, 0], y = [3, 5]\}$



Output: $[-0.6666666666666666, 0]$

User Message: -

How test will be performed: Unit Testing

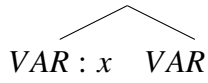
Associated Module: Range Solver

5. test-calculate_divisionPositiveDivisor5

   Type: Functional, Automatic, Integration

   Initial State: New Session

   Input: $/:$, $\{x = [-2, -1], y = [3, 5]\}$



   Output: $[-0.6666666666666666, -0.2]$

   User Message: -

   How test will be performed: Unit Testing

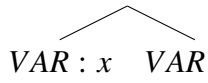
   Associated Module: Range Solver

6. test-calculate_divisionNegativeDivisor1

   Type: Functional, Automatic, Integration

   Initial State: New Session

   Input: $/:$, $\{x = [2, 4], y = [-3, -5]\}$



   Output: $[-0.8, -0.6666666666666666]$

   User Message: -

   How test will be performed: Unit Testing

   Associated Module: Range Solver

7. test-calculate_divisionNegativeDivisor2

   Type: Functional, Automatic, Integration

   Initial State: New Session

   Input: $/ :$, $\{x = [0, 4], y = [-3, -5]\}$

   $VAR : x \quad VAR : y$

   Output: $[-0.8, 0]$

   User Message: -

   How test will be performed: Unit Testing

   Associated Module: Range Solver

8. test-calculate_divisionNegativeDivisor3

   Type: Functional, Automatic, Integration

   Initial State: New Session

   Input: $/ :$, $\{x = [-1, 4], y = [-3, -5]\}$

   $VAR : x \quad VAR : y$

   Output: $[-0.8, 0.2]$

   User Message: -

   How test will be performed: Unit Testing

   Associated Module: Range Solver

9. test-calculate_divisionNegativeDivisor4

   Type: Functional, Automatic, Integration

   Initial State: New Session

   Input: $/ :$, $\{x = [-2, 0], y = [-3, -5]\}$

   $VAR : x \quad VAR : y$

   Output: $[0, 0.4]$

User Message: -

How test will be performed: Unit Testing

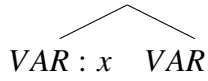
Associated Module: Range Solver

10. test-calculate_divisionNegativeDivisor5

    Type: Functional, Automatic, Integration

    Initial State: New Session

    Input: $/:$, $\{x = [-2, -1], y = [-3, -5]\}$

    Output: [0.33333333333, 0.4]

    User Message: -

    How test will be performed: Unit Testing

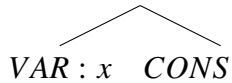
    Associated Module: Range Solver

11. test-calculate_divisionbyzero

    Type: Functional, Automatic, Integration

    Initial State: New Session

    Input: $/:$, $\{x = [2, 4]\}$

    Output: `NULL`

    User Message: Error: An unsupported operation was encountered while solving for the range of the equation (Mixed interval division).

    How test will be performed: Unit Testing

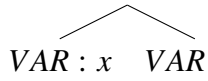
    Associated Module: Range Solver

12. test-calculate_divisionMixedIntervalDivisor

    Type: Functional, Automatic, Integration

Initial State: New Session

Input: $/:$, $\{x = [2, 4], y = [-3, 5]\}$



Output: $NULL$

User Message: Error: An unsupported operation was encountered while solving for the range of the equation (Mixed interval division).

How test will be performed: Unit Testing

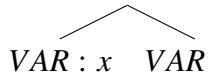
Associated Module: Range Solver

13. test-calculate_divisionMixedIntervalDivisorZero1

    Type: Functional, Automatic, Integration

    Initial State: New Session

    Input: $/:$, $\{x = [2, 4], y = [-3, 0]\}$



    Output: `NULL`

    User Message: Error: An unsupported operation was encountered while solving for the range of the equation (Mixed interval division).

    How test will be performed: Unit Testing

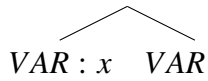
    Associated Module: Range Solver

14. test-calculate_divisionMixedIntervalDivisorZero2

    Type: Functional, Automatic, Integration

    Initial State: New Session

    Input: $/:$, $\{x = [2, 4], y = [0, 3]\}$



    Output: `NULL`

User Message: Error: An unsupported operation was encountered while solving for the range of the equation (Mixed interval division).

How test will be performed: Unit Testing

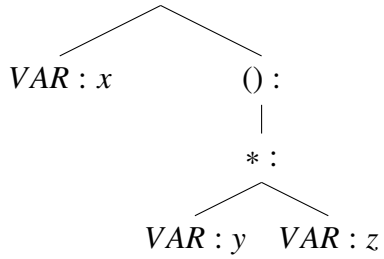
Associated Module: Range Solver

15. test-calculate_mixedDivisorComponent

    Type: Functional, Automatic, Integration

    Initial State: New Session

    Input: $/:$, $\{x = [-2, -1], y = [3, 5], z = [-1, 1]\}$



    Output: $NULL$

    User Message: Error: An unsupported operation was encountered while solving for the range of the equation (Mixed interval division).

    How test will be performed: Unit Testing

    Associated Module: Range Solver

    Note: Ideally, the error message should also output the results calculated up until the division operation (e.g. $[-2, -1]/[-3, 5]$).

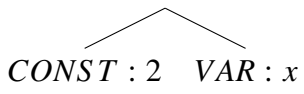
Correctness of Intervals as Exponents

1. test-calculate_intervalAsExponents

    Type: Functional, Automatic, Integration

    Initial State: New Session

    Input: $\wedge:$, $\{x = [2, 4]\}$



Output: [4, 16]

User Message: -

How test will be performed: Unit Testing

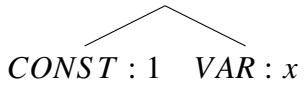
Associated Module: Range Solver

2. test-calculate_intervalAsExponentsInvalidBase

   Type: Functional, Automatic, Integration

   Initial State: New Session

   Input: $\wedge:$, $\{x = [2, 4]\}$

   Output: $NULL$

   User Message: Error: An unsupported operation was encountered while solving for the range of the equation (Exponent base <= 1).

   How test will be performed: Unit Testing

   Associated Module: Range Solver

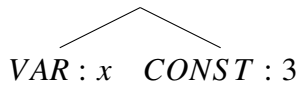
Correctness of Intervals as Base Numbers

1. test-calculate_intervalWithExponent1

   Type: Functional, Automatic, Integration

   Initial State: New Session

   Input: $\wedge:$, $\{x = [2, 4]\}$

   Output: [8, 64]

   User Message: -

   How test will be performed: Unit Testing

   Associated Module: Range Solver

2. test-calculate_intervalWithExponent2

   Type: Functional, Automatic, Integration

   Initial State: New Session

   Input: $\wedge :$, $\{x = [2, 4]\}$

   $VAR : x \quad CONST : 2$

   Output: [4, 16]

   User Message: -

   How test will be performed: Unit Testing

   Associated Module: Range Solver

3. test-calculate_intervalWithExponent3

   Type: Functional, Automatic, Integration

   Initial State: New Session

   Input: $\wedge :$, $\{x = [0, 4]\}$

   $VAR : x \quad CONST : 2$

   Output: [0, 16]

   User Message: -

   How test will be performed: Unit Testing

   Associated Module: Range Solver

4. test-calculate_intervalWithExponent4

   Type: Functional, Automatic, Integration

   Initial State: New Session

   Input: $\wedge :$, $\{x = [-2, 4]\}$

   $VAR : x \quad CONST : 2$

   Output: [0, 16]

User Message: -

How test will be performed: Unit Testing

Associated Module: Range Solver

5. test-calculate_intervalWithExponent5

   Type: Functional, Automatic, Integration

   Initial State: New Session

   Input: $\wedge$: , $\{x = [-4, -2]\}$

   $VAR : x \quad CONST : 2$

   Output: [4, 16]

   User Message: -

   How test will be performed: Unit Testing

   Associated Module: Range Solver

6. test-calculate_intervalWithInvalidExponent1

   Type: Functional, Automatic, Integration

   Initial State: New Session

   Input: $\wedge$: , $\{x = [2, 4]\}$

   $VAR : x \quad CONST : 2$

   Output: [4, 16]

   User Message: Warning: The value provided for the exponent 2.1 is not a natural number. It has been rounded to 2.

   How test will be performed: Unit Testing

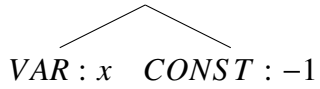
   Associated Module: Range Solver

7. test-calculate_intervalWithInvalidExponent2

   Type: Functional, Automatic, Integration

Initial State: New Session

Input: $\wedge :$, $\{x = [2, 4]\}$



Output: $NULL$

User Message: Error: An unsupported operation was encountered while solving for the range of the equation (Exponent < 0).

How test will be performed: Unit Testing

Associated Module: Range Solver

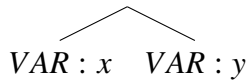
Intervals-Only Exponentiation

1. test-calculate_intervalsOnlyExponentiation

   Type: Functional, Automatic, Integration

   Initial State: New Session

   Input: $\wedge :$, $\{x = [-2, -1], y = [3, 5]\}$

   Output: $NULL$

   User Message: Error: An unsupported operation was encountered while solving for the range of the equation (Exponents).

   How test will be performed: Unit Testing

   Associated Module: Range Solver

5.1.4 Composing Operators

This test suite is designed to determine if the R6 (Recomposing the results from the component equations of $f(V)$) and R7 (Ensuring that the calculated result is represented in closed, interval form) functional requirements, and the associated data constraints from R8 are satisfied.

The tests that follow these assume that the user input tests (5.1.1), input conditioning tests (5.1.2), and component equation calculation tests (5.1.3) have passed.

Operational Precedence Tests

1. test-control_commutativityOfAdditionSubstraction
   Type: Functional, Automatic, System, Regression
   Initial State: New Session
   Input: `x+y-z`, `(x+y)-z`, `x,2,4 \ny,3,5 \n z,2,2`
   Output: `x+y-z = (x+y)-z`
   User Message: -
   How test will be performed: Unit Testing
   Associated Module: Control Flow

2. test-control_commutativityOfMultiplicationDivision
   Type: Functional, Automatic, System, Regression
   Initial State: New Session
   Input: `x*y/z`, `(x*y)/z`, `x,2,4 \ny,3,5 \n z,2,2`
   Output: `x*y/z = (x*y)/z`
   User Message: -
   How test will be performed: Unit Testing
   Associated Module: Control Flow

3. test-control_precedenceOfOperators1
   Type: Functional, Automatic, System, Regression
   Initial State: New Session
   Input: `x+y*z`, `x+(y*z)`, `x,2,4 \ny,3,5 \n z,2,2`
   Output: `x+y*z = x+(y*z)`
   User Message: -
   How test will be performed: Unit Testing
   Associated Module: Control Flow

4. test-control_precedenceOfOperators2
   Type: Functional, Automatic, System, Regression
   Initial State: New Session
   Input: `2^x*y`, `(2^x)*y`, `x,2,4 \ny,3,5`
   Output: `2^x*y = (2^x)*y`
   User Message: -
   How test will be performed: Unit Testing
   Associated Module: Control Flow

5. test-control_precedenceOfOperators3
   Type: Functional, Automatic, System, Regression
   Initial State: New Session
   Input: `x^2*y`, `(x^2)*y`, `x,2,4 \ny,3,5`
   Output: `x^2*y = (x^2)*y`
   User Message: -
   How test will be performed: Unit Testing
   Associated Module: Control Flow

6. test-control_precedenceOfOperators4
   Type: Functional, Automatic, System, Regression
   Initial State: New Session
   Input: `x*(y+z)`, `x,2,4 \ny,3,5\nz,2,2`
   Output: `[10, 28]`
   User Message: -
   How test will be performed: Unit Testing
   Associated Module: Control Flow

7. test-control_precedenceOfOperators5

   Type: Functional, Automatic, System, Regression

   Initial State: New Session

   Input: `2(x+y)^2/3^z`, `x,1,2 \ny,3,4\nz,5,6`

   Output: `[0.0438957475994513, \n0.296296296296296]`

   User Message: -

   How test will be performed: Unit Testing

   Associated Module: Control Flow

8. test-control_precedenceOfOperators6

   Type: Functional, Automatic, System, Regression

   Initial State: New Session

   Input: `(2(x+y)^2)/(3^z)`, `x,1,2 \ny,3,4\nz,5,6`

   Output: `[0.0438957475994513, \n0.296296296296296]`

   User Message: -

   How test will be performed: Unit Testing

   Associated Module: Control Flow

5.1.5 Presentation of Calculations to the User

This test suite is designed to test that the final functional requirement, R9 (Showing the results to the user or an error if a result cannot be found), is satisfied. It is assumed that any inputs that do not produce a viable result have been caught previously and an appropriate error or warning message was presented to the user.

Printing Intervals

1. test-output_printvariableintervalWithName

   Type: Functional, Automatic, System, Regression

   Initial State: New Session

   Input: $x = [2, 4]$

Output: `x = [2, 4]`

User Message: -

How test will be performed: Unit Testing

Associated Module: Output

2. test-output_printvariableintervalNoName

   Type: Functional, Automatic, System, Regression

   Initial State: New Session

   Input: $x = [2, 4]$

   Output: `[2, 4]`

   User Message: -

   How test will be performed: Unit Testing

   Associated Module: Output

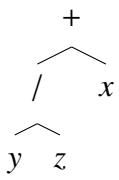
Printing Equation Trees

1. test-output_printequationtree

   Type: Functional, Automatic, System, Regression

   Initial State: New Session

   Input:



   Output:

```
+- {+}
| +- {/}
| | +- {VAR} y
| | +- {VAR} z
| +- {VAR} x
```

   User Message: -

How test will be performed: Unit Testing

Associated Module: Output

5.2 Tests for Non-Functional Requirements

Some of the non-functional requirements can be tested using the same tests as their associated functional requirements. This includes correctness and verifiability (5.1.4), and robustness (applicable at every process stage). This leaves the non-functional requirements of usability and maintainability to test separately.

Due to the intended audience for the initial version of the C^3 tool and the size of the test team, maintainability testing will not be included in this test plan.

5.2.1 Usability

The tests for usability focus on the notation used to represent $f(V)$ and $D(v), v \in V$ and the user's ability to use the tool intuitively without external guidance or instruction. This testing will be completed via a user study (Appendix 7).

Use of Standard Mathematical Notation

1. test-enteringFV

   Type: Non-Functional, Manual, System

   Initial State: New Session

   Condition: The user is given a $f(V)$ to enter into the tool; the $f(V)$ chosen will be taken from the functional requirement tests (Operational Precedence Test, ID: 7)

   Result: The user should be able to enter in the $f(V)$ with no guidance

   How test will be performed: User study (7)

2. test-enteringDv

   Type: Non-Functional, Manual, System

   Initial State: New Session

Condition: The user is given a series of $D(v)$ to enter into the tool; the $D(v)$ chosen will be taken from the same functional requirement tests (5.1) as the $f(V)$ usability test (Test ID: test-enteringFV)

Result: The user should be able to enter in the $D(v)$ with no guidance

How test will be performed: User study (7)

3. test-presentationOutputs

   Type: Non-Functional, Manual, System

   Initial State: End Session

   Condition: The user will be shown the results that the tool produces after it completes its calculations

   Result: The user should be able to find and understand the system's outputs from the GUI with no guidance

   How test will be performed: User study (7)

5.3 Traceability Between Test Cases and Requirements

The purpose of the traceability matrix (Table 1) and graph (Figure 1) is to provide easy references between the test suite and the requirements from the SRS (version 2.0). In the table, the test suites that verify a requirement are marked with an "X" in the requirement's column. In the graph, tests appear at the tail of an arrow and requirements appear at the head.

	R1	R2	R3	R4	R5	R6	R7	R8	R9	NF: Usability
5.1.1	X		X							
5.1.2		X	X	X						
5.1.3					X		X	X		
5.1.4						X	X	X		
5.1.5									X	
5.2.1										X

Table 1: Traceability Matrix Showing the Connections Between Requirements and Test Suites

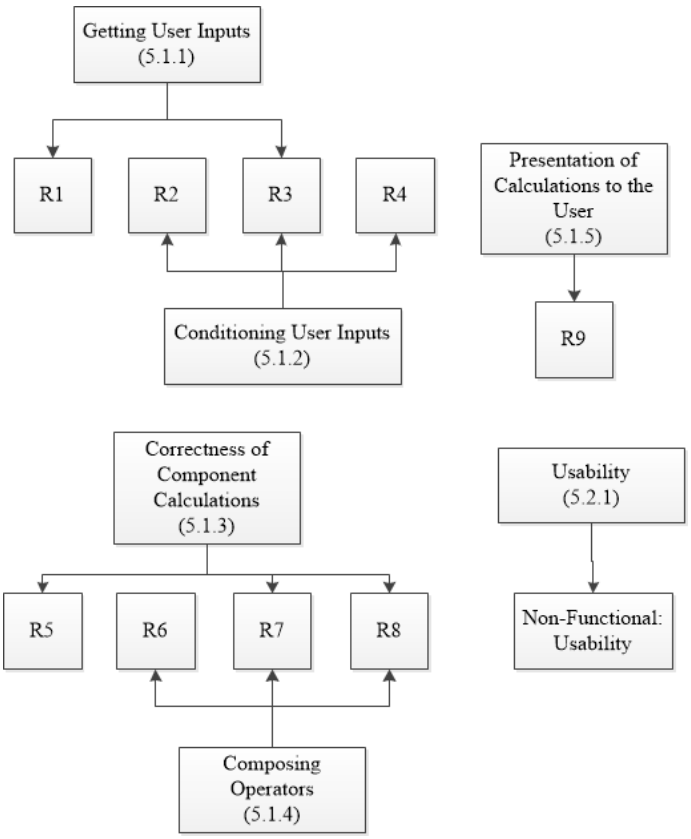


Figure 1: Traceability Graph Showing the Connections Requirements and Test Suites

6 Unit Testing

As shown by the system test description (Section 4), the system and integration testing of the C^3 tool can be divided into several unique steps. However, these tests do not cover all possible code paths such that 100% code coverage can be achieved. The unit tests were created in addition to the functional requirements tests to realize this goal. Unit testing is divided by modules.

6.1 Control Flow Unit Tests

These tests are designed to exercise all code paths in the `ControlFlow.cs` file in the `CompanionCubeCalculator.csproj` project file. They are implemented in the `ControlTests.cs` file in the `UnitTests_CompanionCubeCalculator.csproj` project file. These projects are collected in the `CompanionCubeCalculator.sln` solution file.

1. unittest-controlinit

   Type: Functional, Automatic, Unit

   Initial State: New Session

   Input: *ControlFlow.Initialize*()

   Output: $TRUE$

   User Message: Parser initialized.

   How test will be performed: Unit Testing

2. unittest-controlcondition

   Type: Functional, Automatic, Unit

   Initial State: New Session

   Input: `"x + y"`, $TRUE$

   Output: `"x+y"`

   User Message: -

   How test will be performed: Unit Testing

3. unittest-controlgetfiletypes
   Type: Functional, Automatic, Unit
   Initial State: New Session
   Input: *ControlFlow.GetValidFileTypes*()
   Output: `"*.txt"`
   User Message: -
   How test will be performed: Unit Testing

4. unittest-controlextractvars
   Type: Functional, Automatic, Unit
   Initial State: New Session
   Input: `"x+y"`
   Output: `{ "x", "y" }`
   User Message: -
   How test will be performed: Unit Testing

5. unittest-controlgetvarinfo
   Type: Functional, Automatic, Unit
   Initial State: New Session
   Input: `{ "x+y", "x,2,4\ny,3,5" }`
   Output: `{ { "x", "2", "4" }, { "y", "3", "5" } }`
   User Message: -
   How test will be performed: Unit Testing

6. unittest-controlfile
   Type: Functional, Automatic, Unit
   Initial State: New Session
   Input: `@"TestFiles/test.txt"`

Output: { "x+y", "x,2,4\ny,3,5" }

User Message: -

How test will be performed: Unit Testing

6.2 User Input Unit Tests

These tests are designed to exercise all code paths in the `Input.cs` file in the `CompanionCubeCalculator.csproj` project file. They are implemented in the `InputTests.cs` file in the `UnitTests_CompanionCubeCalculator.csproj` project file. These projects are collected in the `CompanionCubeCalculator.sln` solution file.

File I/O

1. unittest-fileinput

   Type: Functional, Automatic, Unit

   Initial State: New Session

   Input: File containing:

   ```
   x + y
   x,2,4
   y,3,5
   ```

   Output: { ”x+y”, ”x,2,4\ny,3,5” }

   User Message: -

   How test will be performed: Unit Test

2. unittest-fileinputwithequals

   Type: Functional, Automatic, Unit

   Initial State: New Session

   Input: File containing:

   ```
   f(V) = x + y
   x,2,4
   y,3,5
   ```

Output: { ”x+y”, ”x,2,4\ny,3,5” }

User Message: -

How test will be performed: Unit Test

3. unittest-emptyfile

   Type: Functional, Automatic, Unit

   Initial State: New Session

   Input: File containing no information

   Output: $NULL$

   User Message: Error: The file is empty.

   How test will be performed: Unit Test

4. unittest-invalidfiletype

   Type: Functional, Automatic, Unit

   Initial State: New Session

   Input: File with an unsupported extension

   Output: $NULL$

   User Message: Error: Cannot read files of this type.

   How test will be performed: Unit Test

5. unittest-input_noFunctionFile

   Type: Functional, Automatic, Unit

   Initial State: New Session

   Input: A file containing:

   x , 2 , 4
   y , 3 , 5

   Output: $NULL$

   User Message: Error: The first line of the file is not an equation or the equation contains ’,’.

How test will be performed: Unit Test

6. unittest-noFile
   Type: Functional, Automatic, Unit
   Initial State: New Session
   Input: A file that does not exist
   Output: $NULL$
   User Message: Error: The specified file does not exist.
   How test will be performed: Unit Test

7. unittest-badFileInput
   Type: Functional, Automatic, Unit
   Initial State: New Session
   Input: File that cannot be read
   Output: $NULL$
   User Message: Error: Cannot read files of this type.
   How test will be performed: Unit Test

Getters

1. unittest-testgetlinedelimiter
   Type: Functional, Automatic, Unit
   Initial State: New Session
   Input: -
   Output: `"\r\n"`
   User Message: -
   How test will be performed: Unit Testing

2. unittest-testgetfielddelimiter

   Type: Functional, Automatic, Unit

   Initial State: New Session

   Input: -

   Output: `","`

   User Message: -

   How test will be performed: Unit Testing

3. unittest-testgetfiletypes

   Type: Functional, Automatic, Unit

   Initial State: New Session

   Input: -

   Output: `"*.txt"`

   User Message: -

   How test will be performed: Unit Testing

6.3 Interval Data Structure and Interval Conversion Unit Tests

These tests are designed to exercise all code paths in the `IntervalStruct.cs` and `IntervalConversion.cs` files in the `CompanionCubeCalculator.csproj` project file. They are implemented in the `IntervalTests.cs` file in the `UnitTests_CompanionCubeCalculator.csproj` project file. These projects are collected in the `CompanionCubeCalculator.sln` solution file.

Interval Data Structure

1. unittest-intervaldatastructuresetters

   Type: Functional, Automatic, Unit

   Initial State: New Session

   Input: `"x", "3", "4"`, *False*, *False*

Changes: $min = 2, max = 5, leftBoundClosed = TRUE, rightBoundClosed = TRUE$

Output: `"x"`, 2, 5, $True$, $True$

User Message: -

How test will be performed: Unit Testing

2. unittest-intervaldatastructuresetvarname

   Type: Functional, Automatic, Unit

   Initial State: New Session

   Input: `"x", "3", "4"`, $False$, $False$

   Changes: $varName = y$

   Output: `"y"`, 3, 4, $False$, $False$

   User Message: -

   How test will be performed: Unit Testing

Interval Conversion

1. unittest-intervalconversionmissingbounds

   Type: Functional, Automatic, Unit

   Initial State: New Session

   Input: `"x"`

   Output: {}

   User Message: Error: No fields found for variable (Line n). Skipping line.

   How test will be performed: Unit Testing

2. unittest-intervalconversionemptybounds

   Type: Functional, Automatic, Unit

   Initial State: New Session

   Input: `"x,,"`

Output: { }

User Message: Error: No values provided for either interval bound.

How test will be performed: Unit Testing

3. unittest-intervalconversionmissingvarname

   Type: Functional, Automatic, Unit

   Initial State: New Session

   Input: `"3.0,4.0"`

   Output: { }

   User Message: Error: Intervals must have an associated variable name.

   How test will be performed: Unit Testing

4. unittest-intervalconversionemptyvarname

   Type: Functional, Automatic, Unit

   Initial State: New Session

   Input: `",3.0,4.0"`

   Output: { }

   User Message: Error: Intervals must have an associated variable name.

   How test will be performed: Unit Testing

5. unittest-intervalconversiontoomanyfields

   Type: Functional, Automatic, Unit

   Initial State: New Session

   Input: `"x,3,4,5"`

   Output: { }

   User Message: Error: Encountered a variable with more than three fields (Line n). Skipping line.

   How test will be performed: Unit Testing

6.4 Equation Data Structure and Equation Conversion Unit Tests

These tests are designed to exercise all code paths in the `EquationStruct.cs` and `EquationConversion.cs` files in the `CompanionCubeCalculator.csproj` project file. They are implemented in the `EquationTests.cs` file in the `UnitTests_CompanionCubeCalculator.csproj` project file. These projects are collected in the `CompanionCubeCalculator.sln` solution file.

Equation Data Structure

Constructors

1. unittest-equationdatastructurenoop

   Type: Functional, Automatic, Unit

   Initial State: New Session

   Input: `""`, `""`, *NULL*, *NULL*

   Output: `Exception`

   User Message: Error: Equation structures must be assigned an operator during initialization.

   How test will be performed: Unit Testing

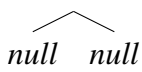
2. unittest-equationdatastructureconstructnulls

   Type: Functional, Automatic, Unit

   Initial State: New Session

   Input: `"+"`, `"x"`, *NULL*, *NULL*

   Output:

   User Message: -

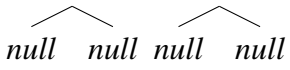
   How test will be performed: Unit Testing

3. unittest-equationdatastructureconstruct

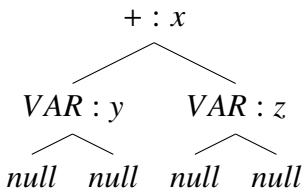
   Type: Functional, Automatic, Unit

   Initial State: New Session

   Input: `"+", "x",` $VAR : y$, $VAR : z$

   Output:

   User Message: -

   How test will be performed: Unit Testing

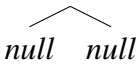
Getters

1. unittest-equationdatastructuresetvarname

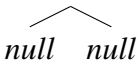
   Type: Functional, Automatic, Unit

   Initial State: New Session

   Input: $+ : x$, y

   Output: $+ : y$

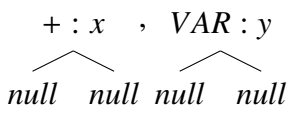
   User Message: -

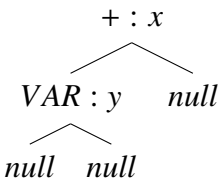
   How test will be performed: Unit Testing

2. unittest-equationdatastructuresetleftop

   Type: Functional, Automatic, Unit

   Initial State: New Session

Input: 

Output: 

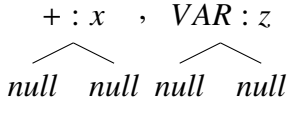
User Message: -

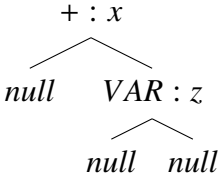
How test will be performed: Unit Testing

3. unittest-equationdatastructuresetrightop

   Type: Functional, Automatic, Unit

   Initial State: New Session

   Input: 

   Output: 

   User Message: -

   How test will be performed: Unit Testing

Equation Conversion

1. unittest-equationconversionconfignoops

   Type: Functional, Automatic, Unit

   Initial State: New Session

   Input: $newOperatorStruct[]\{\}, Solver.GetValidTerminators()$

   Output: $FALSE$

User Message: Error: No operators were passed to the parser.

How test will be performed: Unit Testing

2. unittest-equationconversionconfigunary

   Type: Functional, Automatic, Unit

   Initial State: New Session

   Input: $newOperatorStruct[]\{newOperatorStruct(\text{"}-\text{"}, 5, true, false, false, false)\}$, $Solver.GetValidTerminators()$

   Output: $TRUE$

   User Message: -

   How test will be performed: Unit Testing

3. unittest-equationconversionconfigbinary

   Type: Functional, Automatic, Unit

   Initial State: New Session

   Input: $newOperatorStruct[]\{newOperatorStruct(\text{"}-\text{"}, 5, false, true, false, false)\}$, $Solver.GetValidTerminators()$

   Output: $TRUE$

   User Message: -

   How test will be performed: Unit Testing

4. unittest-equationconversionconfigternary

   Type: Functional, Automatic, Unit

   Initial State: New Session

   Input: $newOperatorStruct[]\{newOperatorStruct(\text{"\&"}, 5, false, false, true, false)\}$, $Solver.GetValidTerminators()$

   Output: $FALSE$

   User Message: Error: The parser cannot process the & operator.

   How test will be performed: Unit Testing

5. unittest-equationconversionconfigunbalancedleftterm

   Type: Functional, Automatic, Unit

   Initial State: New Session

   Input: $Solver.GetValidOperators()$, $newstring[]\{"(", ""\}$

   Output: $FALSE$

   User Message: Error: An unbalanced left terminator token was encountered (().

   How test will be performed: Unit Testing

6. unittest-equationconversionconfigunbalancedrightterm

   Type: Functional, Automatic, Unit

   Initial State: New Session

   Input: $Solver.GetValidOperators()$, $newstring[]\{"", ")"\}$

   Output: $FALSE$

   User Message: Error: An unbalanced right terminator token was encountered ()).

   How test will be performed: Unit Testing

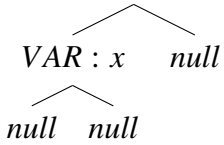
7. unittest-equationconversionparseunary

   Type: Functional, Automatic, Unit

   Initial State: New Session

   Input: `"-x"`, $newOperatorStruct(" - ", 5, true, false, false, false)$

   Output:



   User Message: -

   How test will be performed: Unit Testing

8. unittest-equationconversion_missingFunctionValue1

   Type: Functional, Automatic, Unit

   Initial State: New Session

   Input: `"x+"`

   Output: *NULL*

   User Message: Error: Could not find the end of the equation.

   How test will be performed: Unit Test

9. unittest-equationconversion_missingFunctionValue2

   Type: Functional, Automatic, Unit

   Initial State: New Session

   Input: `"*x"`

   Output: *NULL*

   User Message: Error: Unrecognized sequence encountered during Atomic Equation parsing. Remaining equation = *x.

   How test will be performed: Unit Test

10. unittest-equationconversion_missingFunctionValue3

    Type: Functional, Automatic, Unit

    Initial State: New Session

    Input: `"x+*y"`

    Output: *NULL*

    User Message: Error: Unrecognized sequence encountered during Atomic Equation parsing. Remaining equation = *y.

    How test will be performed: Unit Test

6.5 Variable Consolidation Unit Tests

These tests are designed to exercise all code paths in the `Consolidate.cs` file in the `CompanionCubeCalculator.csproj` project file. They are implemented in the `VariableConsolidationTests.cs` file in the `UnitTests_CompanionCubeCalculator.csproj` project file. These projects are collected in the `CompanionCubeCalculator.sln` solution file.

1. unittest-consolidatetestinit

   Type: Functional, Automatic, Integration

   Initial State: New Session

   Input: -

   Output: $TRUE$

   User Message: -

   How test will be performed: Unit Testing

2. unittest-consolidatetestinit

   Type: Functional, Automatic, Integration

   Initial State: New Session

   Input: -

   Output: $TRUE$

   User Message: -

   How test will be performed: Unit Testing

3. unittest-consolidatenoops

   Type: Functional, Automatic, Integration

   Initial State: New Session

   Input: *ConvertAndCheckInputs*(`"x+y"`, `"x,2,3 \ny,4,5"`, {}, *Solver.GetValidTerminators*())

   Output: $successCode = -1$

   User Message: Error: No operators were passed to the parser.

How test will be performed: Unit Testing

4. unittest-consolidatenorightterm

   Type: Functional, Automatic, Integration

   Initial State: New Session

   Input: *ConvertAndCheckInputs*("x+y", "x,2,3 \ny,4,5", *Solver.GetValidOperators*(), {{"("}})

   Output: $successCode = -1$

   User Message: Error: An unbalanced left terminator token was encountered ("(")

   How test will be performed: Unit Testing

5. unittest-consolidatenoleftterm

   Type: Functional, Automatic, Integration

   Initial State: New Session

   Input: *ConvertAndCheckInputs*("x+y", "x,2,3 \ny,4,5", *Solver.GetValidOperators*(), {{")"}})

   Output: $successCode = -1$

   User Message: Error: An unbalanced right terminator token was encountered (")")

   How test will be performed: Unit Testing

6. unittest-consolidatesimpleinputs

   Type: Functional, Automatic, Integration

   Initial State: New Session

   Input: *ConvertAndCheckInputs*("x+y", "x,2,3 \ny,4,5", *Solver.GetValidOperators*(), *Solver.GetValidTerminators*())

   Output: $\overset{+}{\underset{x \quad y}{\wedge}}$, { $x = [2, 3]$, $y = [4, 5]$ }

User Message: -

How test will be performed: Unit Testing

7. unittest-consolidateextractvariables

   Type: Functional, Automatic, Integration

   Initial State: New Session

   Input: *ConvertAndCheckInputs*("x+y", "x,2,3 \ny,4,5", *Solver.GetValidOperators*(), *Solver.GetValidTerminators*());
   Consolidate.GetIntervalStructList()

   Output: { $x = [2, 3]$, $y = [4, 5]$ }

   User Message: -

   How test will be performed: Unit Testing

8. unittest-consolidateincompleteequation

   Type: Functional, Automatic, Integration

   Initial State: New Session

   Input: "", "x,2,3\n"

   Output: *successCode* $= -1$

   User Message: Error: Unrecognized sequence encountered during Atomic Equation parsing. Remaining equation =

   How test will be performed: Unit Testing

6.6 Operator Data Structure and Range Solver Unit Tests

These tests are designed to exercise all code paths in the `OperatorStruct.cs` and `Solver.cs` files in the `CompanionCubeCalculator.csproj` project file. They are implemented in the `SolverTests.cs` file in the `UnitTests_CompanionCubeCalculator.csproj` project file. These projects are collected in the `CompanionCubeCalculator.sln` solution file.

Operator Data Structure

1. unittest-operatordatastructureconstructor

   Type: Functional, Automatic, Unit

   Initial State: New Session

   Input: "+", 1, $false, true, false, true$

   Output: $OperatorStruct$("+", 1, $false, true, false, true$)

   User Message: -

   How test will be performed: Unit Testing

2. unittest-operatordatastructuremissingsym

   Type: Functional, Automatic, Unit

   Initial State: New Session

   Input: "", 1, $false, true, false, true$

   Output: `Exception`

   User Message: Error: Cannot have an operator with no representative symbol.

   How test will be performed: Unit Testing

3. unittest-operatordatastructurelowprecedence

   Type: Functional, Automatic, Unit

   Initial State: New Session

   Input: "+", 0, $false, true, false, true$

   Output: `Exception`

   User Message: Error: Precedence values must be greater than 0.

   How test will be performed: Unit Testing

4. unittest-operatordatastructureoverloadedtype1

   Type: Functional, Automatic, Unit

Initial State: New Session

Input: "+", 1, $true$, $true$, $false$, $true$

Output: `Exception`

User Message: Error: An operator cannot be overloaded to be unary, binary, and ternary.

How test will be performed: Unit Testing

5. unittest-operatordatastructureoverloadedtype2

   Type: Functional, Automatic, Unit

   Initial State: New Session

   Input: "+", 1, $false$, $true$, $true$, $true$

   Output: `Exception`

   User Message: Error: An operator cannot be overloaded to be unary, binary, and ternary.

   How test will be performed: Unit Testing

6. unittest-operatordatastructureoverloadedtype3

   Type: Functional, Automatic, Unit

   Initial State: New Session

   Input: "+", 1, $true$, $false$, $true$, $true$

   Output: `Exception`

   User Message: Error: An operator cannot be overloaded to be unary, binary, and ternary.

   How test will be performed: Unit Testing

7. unittest-operatordatastructurenotype

   Type: Functional, Automatic, Unit

   Initial State: New Session

   Input: "+", 1, $false$, $false$, $false$, $true$

Output: `Exception`

User Message: Error: Operators must be assigned a number of operands type.

How test will be performed: Unit Testing

Range Solver

1. unittest-solvernoequation

   Type: Functional, Automatic, Integration

   Initial State: New Session

   Input: $null, x = [1, 2]$

   Output: `Exception`

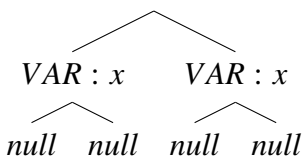
   User Message: Error: No information was provided for the equation.

   How test will be performed: Unit Testing

2. unittest-solverunknownop

   Type: Functional, Automatic, Integration

   Initial State: New Session

   Input: 

   Output: $NULL$

   User Message: Error: An unsupported operation was encountered while solving for the range of the equation (Unknown operator).

   How test will be performed: Unit Testing

3. unittest-solvermissingintervals

   Type: Functional, Automatic, Integration

Initial State: New Session

Input: $VAR : x$, {}

$$\begin{array}{c} VAR : x \\ \diagup \quad \diagdown \\ null \quad null \end{array}$$

Output: $NULL$

User Message: Error: Could not find an associated interval for variable x.

How test will be performed: Unit Testing

4. unittest-solverconstantfunction

   Type: Functional, Automatic, Integration

   Initial State: New Session

   Input: $CONST : 42, \{\}$

   $$\begin{array}{c} CONST : 42 \\ \diagup \quad \diagdown \\ null \quad null \end{array}$$

   Output: $[42, 42]$

   User Message: -

   How test will be performed: Unit Testing

5. unittest-solvervariablefunction

   Type: Functional, Automatic, Integration

   Initial State: New Session

   Input: $VAR : x$, $\{x = [2, 3]\}$

   $$\begin{array}{c} VAR : x \\ \diagup \quad \diagdown \\ null \quad null \end{array}$$

   Output: $[2, 3]$

   User Message: -

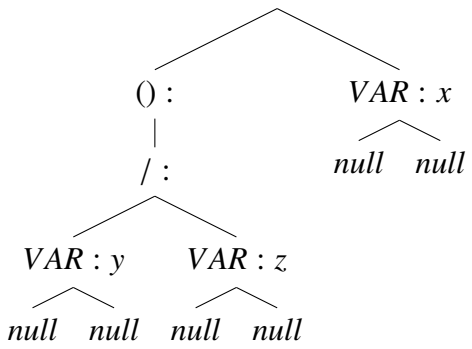
   How test will be performed: Unit Testing

6. unittest-solveradditionleftsidenull

   Type: Functional, Automatic, Integration

Initial State: New Session

Input: $+:$,



$\{x = [2, 4], y = [-3, 5], z = [-1, 1]\}$

Output: $NULL$

User Message: Error: An unsupported operation was encountered while solving for the range of the equation (Mixed interval division).

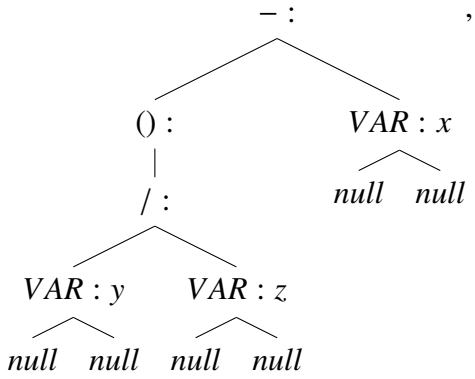
How test will be performed: Unit Testing

7. unittest-solversubtractionleftsidenull

   Type: Functional, Automatic, Integration

   Initial State: New Session

   Input: $-:$,



   $\{x = [2, 4], y = [-3, 5], z = [-1, 1]\}$

   Output: $NULL$

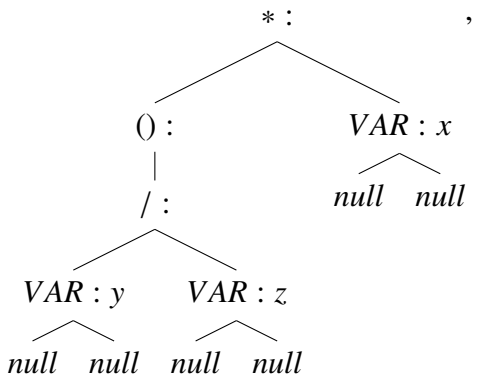
   User Message: Error: An unsupported operation was encountered while solving for the range of the equation (Mixed interval division).

How test will be performed: Unit Testing

8. unittest-solvermultiplicationleftsidenull

   Type: Functional, Automatic, Integration

   Initial State: New Session

   Input: 

   ,
   $\{x = [2, 4], y = [-3, 5], z = [-1, 1]\}$

   Output: $NULL$

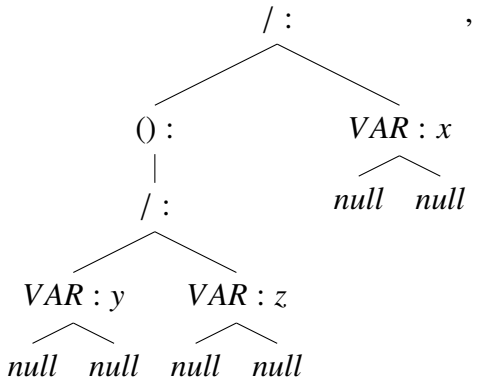
   User Message: Error: An unsupported operation was encountered while solving for the range of the equation (Mixed interval division).

   How test will be performed: Unit Testing

9. unittest-solverdivisionleftsidenull

   Type: Functional, Automatic, Integration

   Initial State: New Session

   Input: 

   ,

$\{x = [2, 4], y = [-3, 5], z = [-1, 1]\}$

Output: $NULL$

User Message: Error: An unsupported operation was encountered while solving for the range of the equation (Mixed interval division).

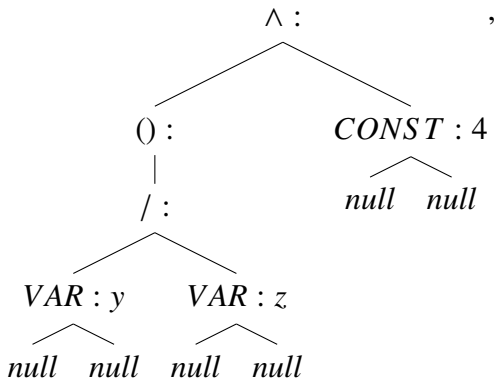
How test will be performed: Unit Testing

10. unittest-solverexponentleftsidenull

    Type: Functional, Automatic, Integration

    Initial State: New Session

    Input:



    $\{y = [-3, 5], z = [-1, 1]\}$

    Output: $NULL$

    User Message: Error: An unsupported operation was encountered while solving for the range of the equation (Mixed interval division).

    How test will be performed: Unit Testing

6.7 Output Unit Tests

These tests are designed to exercise all code paths in the `Output.cs` file in the `CompanionCubeCalculator.csproj` project file. They are implemented in the `OutputTests.cs` file in the `UnitTests_CompanionCubeCalculator.csproj` project file. These projects are collected in the `CompanionCubeCalculator.sln` solution file.

1. unittest-outputopenstructure
   Type: Functional, Automatic, Integration
   Initial State: New Session
   Input: $x = (2, 4), true$
   Output: `x = (2, 4)`
   User Message: -
   How test will be performed: Unit Testing

2. unittest-outputclosedleftstructure
   Type: Functional, Automatic, Integration
   Initial State: New Session
   Input: $x = [2, 4), true$
   Output: `x = [2, 4)`
   User Message: -
   How test will be performed: Unit Testing

3. unittest-outputclosedrightstructure
   Type: Functional, Automatic, Integration
   Initial State: New Session
   Input: $x = (2, 4], true$
   Output: `x = (2, 4]`
   User Message: -
   How test will be performed: Unit Testing

4. unittest-outputconstwithname
   Type: Functional, Automatic, Integration
   Initial State: New Session
   Input: $CONST : 3, true$

Output: `CONST: 3`

User Message: -

How test will be performed: Unit Testing

5. unittest-outputconstnoname

   Type: Functional, Automatic, Integration

   Initial State: New Session

   Input: $CONST : 3, false$

   Output: `CONST: 3`

   User Message: -

   How test will be performed: Unit Testing

6. unittest-outputlongresult

   Type: Functional, Automatic, Integration

   Initial State: New Session

   Input: $x = [0.333333333333, 0.66666666666666], false$

   Output: `[0.333333333333, \n 0.66666666666666]`

   User Message: -

   How test will be performed: Unit Testing

7 Appendix: Usability Testing

An informal user study was conducted as part of the Companion Cube Calculator test suite, where participants were recruited from the examiner's lab. Participants were sent a compressed file containing the program, user manual, and instructions and asked to complete it on their own time. The instructions file contained a URL to the associated study questionnaire.

The instructions that were sent to participants can be found at:

https://github.com/GenevaS/CAS741/tree/master/Doc/TestReport/UserStudy

7.1 Tasks

User study participants were asked to complete two tasks given the information:

$$\frac{2(x+y)^2}{3^z}$$

where

$$x = [1, 2], y = [3, 4], z = [5, 6]$$

The first task was to input the information directly into the GUI and the second task was to load the information into the program via a file.

Participants were free to access the program user guide as they saw fit.

7.2 Questionnaire

This survey were designed to accompany the user study. The questionnaire was distributed as a Google Form. All Likert scale questions were from 1 to 5, where 1 is "Strongly Disagree" and 5 is "Strongly Agree".

7.2.1 Test Results (Round values as necessary)

1. (Long Answer) What range value did you calculate in Task 1?
2. (Long Answer) What was the resulting equation tree from Task 1?
3. (Long Answer) What range value did you calculate in Task 2?
4. (Long Answer) What was the resulting equation tree from Task 2?

7.2.2 Software Usability

Task 1: Entering values directly into the GUI

1. (Likert Scale)I found it easy to find the input mechanism for entering the equation into the GUI.
2. (Likert Scale) I found it easy to use the function input mechanism.
3. (Likert Scale) I found it easy to find the input mechanism for entering variable domains into the GUI.
4. (Likert Scale) I found it easy to use the variable domain input mechanism.

Task 2: Loading values into the program from a file

1. (Likert Scale) I found it easy to find the option to load from a file.

Understanding the Results

1. (Likert Scale) I found it easy to find the calculated range in the GUI.
2. (Likert Scale) I found it easy to understand the calculated range.
3. (Likert Scale) I found it easy to understand the sequence of steps required to use the software.
4. (Long answer) Describe any interactions with the program that confused you during the tasks.
5. (Long answer) Do you have any suggestions for improving the process flow of the tool?