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

Geneva Smith

November 13, 2017

1 Revision History

Date	Version	Notes
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

List of Tables

List of Figures

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 1.1.1), including a description of automated testing approach, tools, and black box 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's SRS documentation (version 1.1.1). 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. The tool has not yet been built, so the plan not include implementation-specific tests. Instead, this plan will outline black box tests that abstracts the tool into modules based on the requirements identified in the SRS documentation.

The purpose of this plan is to guide the modularization of the C^3 tool design so that it is easy to test and maintain. It also forms the foundation of the kinds of behaviours 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 how unit testing will be implemented when the internal workings of the C^3 tool are established.

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 function given the domains of the function's component variables. Function 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. If a result cannot be found, the tool is expected to communicate the reason back to the user. Calculations must be accurate within an error range of the host machine's floating point error. 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 1.1.1). 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 tools work flow. Since these behaviours are internal to the tool with no user guidance, these behaviours can be tested automatically using a series of black box unit tests. Some of the non-functional requirements such as reliability and robustness (Section 5.2), can also be tested using automated approaches because they focus on floating-point error rates and 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 gathering user inputs and ending with the system's outputs. Outputs can be messages informing the user of erroneous behaviours or malformed inputs, and calculated result 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.

- **Automated GUI testing**
  This version [The selected version —SS] of Visual Studio allows developers to record a series of interactions with a GUI that can be played multiple times for testing. This can help save time by making traditionally manual test cases into automated ones. Depending on the complexity of the GUI, this tool might not be used.

- **Code coverage tools that can be run "live" as code is written**
  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. The IDE can be configured so that this code coverage check is run automatically while the program is being written.

4.5 Non-Testing Based Verification

There are no non-testing based verification approaches in this test plan due to time and team constraints.

5 System Test Description

The system testing of the C^3 tool will focus on inspecting and conditioning user inputs and ensuring that the C^3 tool is able to produce descriptive outputs for both valid and invalid inputs.

The goal of user input inspection and conditioning tests is to be sure that the C^3 tool is able to identify and reject values that violate the assumptions. If the tool determines that the values do not violate the assumptions, it should be able to convert those values into the internal representations identified in the SRS document. For the purposes of this test plan, it is assumed that the user function is represented internally as a parse tree.

Even if the user inputs are validated and conditioned correctly, it is still possible for the C^3 tool to encounter invalid operations in an intermediary step while calculating a result. This makes it necessary to create a series of tests to determine how the tool will react to both supported and unsupported operations.

This test plan is meant to be executed in stages. The first stage is a white-box unit test of the functionality required to get raw inputs from the user. The subsequent stages are both integration and unit tests, where the test checks that the behaviour produced by the new modules is both working as expected and can

communicate with the modules above it. The final tests in this chain, which show the calculations of the C^3 tool, form the basis of regression testing and contains mathematical functions that will be incorrectly processed if the tool does not parse them correctly.

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 group assumes that the groups preceding it have passed successfully, adding to the confidence in the product.

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 is satisfied and its associated data constraints from R3.

The initial test cases (User Inputs (As Expected)) are the white-box unit tests that start the integration testing. The tests that follow these assume that these tests have passed.

User Inputs (As Expected)

1. test-directinput

   Type: Functional, Automatic, Unit

   Initial State: New Session

   Input: $f(V) = x + y, x = [2, 4], y = [3, 5]$

   Output: Input received from direct input

   How test will be performed: Unit Test

2. test-fileinput

   Type: Functional, Automatic, Unit

   Initial State: New Session

   Input: File containing: $f(V) = x + y, x = [2, 4], y = [3, 5]$

   Output: Input received from file

   How test will be performed: Unit Test

User Inputs (Bad File I/O)

1. test-badFileInput

   Type: Functional, Automatic, Unit

   Initial State: New Session

   Input: File that cannot be read

   Output: Error – File cannot be read

   How test will be performed: Unit Test

User Inputs (Function is a Constant Value)

1. test-input_functionAsConstant

   Type: Functional, Automatic, Integration

   Initial State: New Session

   Input: $f(V) = 34$

   Output: Warning – Function $f(V)$ does not contain any variables

   How test will be performed: Unit Test

User Inputs (Extraneous Information)

1. test-input_variableNotInFunction

   Type: Functional, Automatic, Integration

Initial State: New Session

Input: $f(V) = x + y, x = [2, 4], y = [3, 5], z = [6, 7]$

Output: Warning – Function $f(V)$ does not contain name z found in variable list ->ignoring domain of z

How test will be performed: Unit Test

User Inputs (Missing Values)

1. test-input_noFunction

   Type: Functional, Automatic, Integration

   Initial State: New Session

   Input: $x = [2, 4], y = [3, 5]$

   Output: Error – Could not find function $f(V)$ with variables x, y

   How test will be performed: Unit Test

2. test-input_noDomain

   Type: Functional

   Initial State: New Session

   Input: $f(X) = x + y, x = [2, 4]$

   Output: Error – No domain for variable y

   How test will be performed: Unit Test

User Inputs (Incomplete Values)

1. test-input_missingFunctionValue1

   Type: Functional, Automatic, Integration

   Initial State: New Session

   Input: $f(V) = x+, x = [2, 4], y = [3, 5]$

   Output: Error – Function $f(V)$ is missing an operand for "+" ->domain list contains unreferenced variable y

How test will be performed: Unit Test

2. test-input_missingFunctionValue2
   Type: Functional, Automatic, Integration
   Initial State: New Session
   Input: $f(V) = x + *y$, $x = [2, 4], y = [3, 5]$
   Output: Error – Function $f(V)$ is missing an operand between + and $*$
   How test will be performed: Unit Test

3. test-input_missingMinDomainValue
   Type: Functional, Automatic, Integration
   Initial State: New Session
   Input: $f(V) = x + y$, $x = [, 4], y = [3, 5]$
   Output: Error – Missing min domain value for x
   How test will be performed: Unit Test

4. test-input_missingMaxDomainValue
   Type: Functional, Automatic, Integration
   Initial State: New Session
   Input: $\{f(V) = x + y, x = [2, 4], y = [3,]\}$, $\{f(V) = x + y, x = [2, 4], y = [3]\}$
   Output: Error – Missing max domain value for y
   How test will be performed: Unit Test

User Inputs (Non-Numerical Value in Domains)

1. test-input_nonNumberInDomain
   Type: Functional, Automatic, Integration
   Initial State: New Session
   Input: $f(V) = x + y$, $x = [2, 4], y = [a, 5]$

Output: Error – Non-numerical value found in the domain for variable y

How test will be performed: Unit Test

Variable Names

1. test-input_simpleVariableName

   Type: Functional, Automatic, Integration

   Initial State: New Session

   Input: $f(V) = x + y$, $x = [2, 4], y = [3, 5]$

   Output: N/A

   How test will be performed: Unit Test

2. test-input_multiCharacterVariableName1

   Type: Functional, Automatic, Integration

   Initial State: New Session

   Input: $f(V) = x1 + y$, $x1 = [2, 4], y = [3, 5]$

   Output: N/A

   How test will be performed: Unit Test

3. test-input_multiCharacterVariableName2

   Type: Functional, Automatic, Integration

   Initial State: New Session

   Input: $f(V) = x\_ + y$, $x\_ = [2, 4], y = [3, 5]$

   Output: N/A

   How test will be performed: Unit Test

4. test-input_multiCharacterVariableName3

   Type: Functional, Automatic, Integration

   Initial State: New Session

Input: $f(V) = x\_1 + y$, $x\_1 = [2, 4], y = [3, 5]$
Output: N/A
How test will be performed: Unit Test

5. test-input_multiCharacterVariableName4
   Type: Functional, Automatic, Integration
   Initial State: New Session
   Input: $f(V) = x\_i + y$, $x\_i = [2, 4], y = [3, 5]$
   Output: N/A
   How test will be performed: Unit Test

6. test-input_multiCharacterVariableName5
   Type: Functional, Automatic, Integration
   Initial State: New Session
   Input: $f(V) = x'' + y$, $x'' = [2, 4], y = [3, 5]$
   Output: N/A
   How test will be performed: Unit Test

7. test-input_multiCharacterVariableName6
   Type: Functional, Automatic, Integration
   Initial State: New Session
   Input: $f(V) = xy + y$, $xy = [2, 4], y = [3, 5]$
   Output: N/A
   How test will be performed: Unit Test

8. test-input_multiCharacterVariableName7
   Type: Functional, Automatic, Integration
   Initial State: New Session

Input: $f(V) = xy$, $x = [2, 4], y = [3, 5]$

Output: Warning – Found variable xy that does not exist in the domain list for $f(V)$ ->expanding to multiplication $x * y$

How test will be performed: Unit Test

9. test-input_multiCharacterVariableName8

   Type: Functional, Automatic, Integration

   Initial State: New Session

   Input: $f(V) = x* + y$, $x* = [2, 4], y = [3, 5]$

   Output: Error – Variable names cannot contain the characters $\{+, -, *, \hat{}, (,)\}$

   How test will be performed: Unit Test

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.

The tests that follow these assume that the user input tests (5.1.1) have passed.

Unexpected Domain Ordering

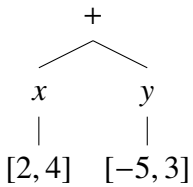
1. test-parse_domainOrder

   Type: Functional, Automatic, Integration

   Initial State: New Session

   Input: $f(V) = x + y$, $x = [2, 4], y = [3, -5]$

   Output: Warning – Domain for variable y is ordered highest to lowest; re-ordering values to be smallest to highest



How test will be performed: Unit Testing

Addition, Subtraction, and Multiplication

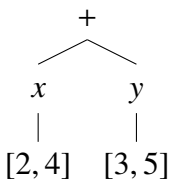
1. test-parse_addition

   Type: Functional, Automatic, Integration

   Initial State: New Session

   Input: $f(V) = x + y$, $x = [2, 4], y = [3, 5]$

   Output:

   How test will be performed: Unit Testing

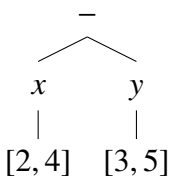
2. test-parse_subtraction

   Type: Functional, Automatic, Integration

   Initial State: New Session

   Input: $f(V) = x - y$, $x = [2, 4], y = [3, 5]$

   Output:

   How test will be performed: Unit Testing

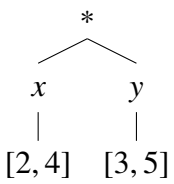
3. test-parse_multiplication1

   Type: Functional, Automatic, Integration

   Initial State: New Session

   Input: $f(V) = x * y$, $x = [2, 4], y = [3, 5]$

Output:



How test will be performed: Unit Testing

Division

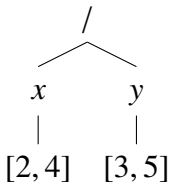
1. test-parse_divisionPositiveDivisor

   Type: Functional, Automatic, Integration

   Initial State: New Session

   Input: $f(V) = x/y$, $x = [2, 4], y = [3, 5]$

   Output:



   How test will be performed: Unit Testing

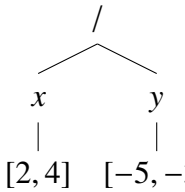
2. test-parse_divisionNegativeDivisor

   Type: Functional, Automatic, Integration

   Initial State: New Session

   Input: $f(V) = x/y$, $x = [2, 4], y = [-5, -3]$

   Output:



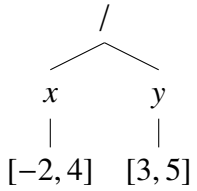
   How test will be performed: Unit Testing

3. test-parse_divisionMixedIntervalDivisor
   Type: Functional, Automatic, Integration
   Initial State: New Session
   Input: $f(V) = x/y$, $x = [2, 4], y = [-3, 5]$
   Output: Error – Cannot perform division with a mixed interval divisor
   How test will be performed: Unit Testing

4. test-parse_divisionMixedIntervalDivisorZero1
   Type: Functional, Automatic, Integration
   Initial State: New Session
   Input: $f(V) = x/y$, $x = [2, 4], y = [-3, 0]$
   Output: Error – Cannot perform division with a mixed interval divisor
   How test will be performed: Unit Testing

5. test-parse_divisionMixedIntervalDivisorZero2
   Type: Functional, Automatic, Integration
   Initial State: New Session
   Input: $f(V) = x/y$, $x = [2, 4], y = [0, 3]$
   Output: Error – Cannot perform division with a mixed interval divisor
   How test will be performed: Unit Testing

6. test-parse_divisionMixedIntervalQuotient
   Type: Functional, Automatic, Integration
   Initial State: New Session
   Input: $f(V) = x/y$, $x = [-2, 4], y = [3, 5]$
   Output:



How test will be performed: Unit Testing

Intervals as Exponents

1. test-parse_intervalAsExponents

   Type: Functional, Automatic, Integration

   Initial State: New Session

   Input: $f(V) = 2^x$, $x = [2, 4]$

   Output: B^x

   $2 \quad x$

   $[2, 4]$

   How test will be performed: Unit Testing

2. test-parse_intervalAsExponentsInvalidBase

   Type: Functional, Automatic, Integration

   Initial State: New Session

   Input: $f(V) = 1^x$, $x = [2, 4]$

   Output: Error – Cannot perform operation with an interval exponent when the base number $(B) \leq 1$

   How test will be performed: Unit Testing

Intervals as Base Numbers

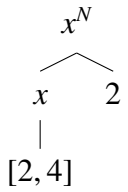
1. test-parse_intervalWithExponent

   Type: Functional, Automatic, Integration

   Initial State: New Session

   Input: $f(V) = x^2$, $x = [2, 4]$

Output:



How test will be performed: Unit Testing

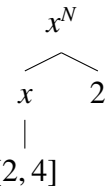
2. test-parse_intervalWithInvalidExponent1

   Type: Functional, Automatic, Integration

   Initial State: New Session

   Input: $f(V) = x^{2.1}$, $x = [2, 4]$

   Output: Warning – Cannot perform operation with an exponent that is not a whole number; the exponent has been rounded to 2



   How test will be performed: Unit Testing

3. test-parse_intervalWithInvalidExponent2

   Type: Functional, Automatic, Integration

   Initial State: New Session

   Input: $f(V) = x^{-1}$, $x = [2, 4]$

   Output: Error – Cannot perform operation exponentiation with an exponent $(N) < 0$

   How test will be performed: Unit Testing

Intervals-Only Exponentiation

1. test-parse_intervalsOnlyExponentiation

   Type: Functional, Automatic, Integration

Initial State: New Session

Input: $f(V) = x^y$, $x = [2, 4], y = [3, 5]$

Output: Error – Cannot perform exponentiation when the base number (B) and the exponent (N) are intervals

How test will be performed: Unit Testing

Constant Values

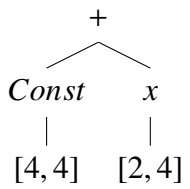
1. test-parse_constantValue1

   Type: Functional, Automatic, Integration

   Initial State: New Session

   Input: $f(V) = 4 + x$, $x = [2, 4]$

   Output:

   How test will be performed: Unit Testing

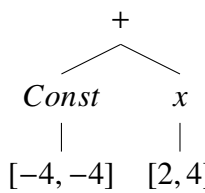
2. test-parse_constantValue2

   Type: Functional, Automatic, Integration

   Initial State: New Session

   Input: $f(V) = -4 + x$, $x = [2, 4]$

   Output:

   How test will be performed: Unit Testing

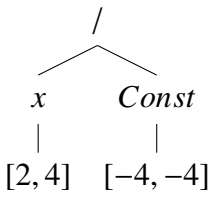
3. test-parse_constantValue3

   Type: Functional, Automatic, Integration

   Initial State: New Session

   Input: $f(V) = x/-4$, $x = [2, 4]$

   Output:



   How test will be performed: Unit Testing

4. test-parse_constantValue4

   Type: Functional, Automatic, Integration

   Initial State: New Session

   Input: $f(V) = x/0$, $x = [2, 4]$

   Output: Error – No support for division by zero

   How test will be performed: Unit Testing

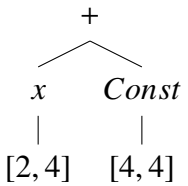
5. test-parse_constantValue5

   Type: Functional, Automatic, Integration

   Initial State: New Session

   Input: $f(V) = x + 4$, $x = [2, 4]$

   Output:



   How test will be performed: Unit Testing

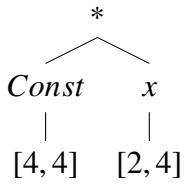
6. test-parse_implicitMultiplication

   Type: Functional, Automatic, Integration

   Initial State: New Session

   Input: $f(V) = 4x$, $x = [2, 4]$

   Output:



   How test will be performed: Unit Testing

Precedence of Brackets

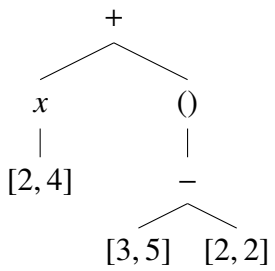
1. test-parse_brackets1

   Type: Functional, Automatic, Integration

   Initial State: New Session

   Input: $f(V) = x + (y - z)$, $x = [2, 4], y = [3, 5], z = [2, 2]$

   Output:



   How test will be performed: Unit Testing

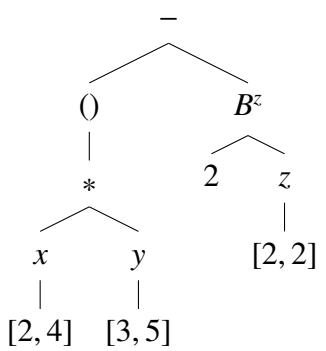
2. test-parse_brackets2

   Type: Functional, Automatic, Integration

   Initial State: New Session

   Input: $f(V) = (x * y) - 2^z$, $x = [2, 4], y = [3, 5], z = [2, 2]$

Output:



How test will be performed: Unit Testing

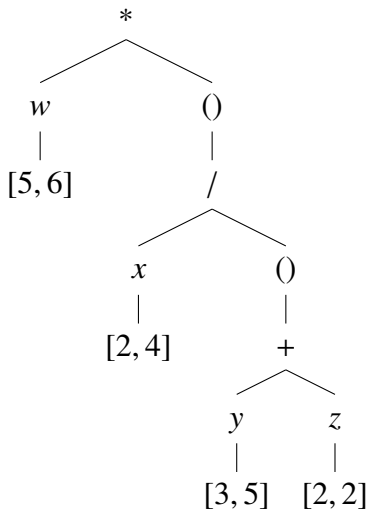
3. test-parse_brackets3

   Type: Functional, Automatic, Integration

   Initial State: New Session

   Input: $f(V) = w * (x/(y + z))$, $w = [5, 6], x = [2, 4], y = [3, 5], z = [2, 2]$

   Output:



   How test will be performed: Unit Testing

Open Brackets

1. test-parse_openLeftBracket

Type: Functional, Automatic, Integration

Initial State: New Session

Input: $f(V) = x + y - z)$, $x = [2, 4], y = [3, 5], z = [2, 2]$

Output: Error – Missing "(" in $f(V)$

How test will be performed: Unit Testing

2. test-parse_openRightBracket

   Type: Functional, Automatic, Integration

   Initial State: New Session

   Input: $f(V) = x + (y - z$, $x = [2, 4], y = [3, 5], z = [2, 2]$

   Output: Error – Missing ")" in $f(V)$

   How test will be performed: Unit Testing

Operational Precedence Tests

1. test-parse_commutativityOfAdditionSubstraction

   Type: Functional, Automatic, System, Regression

   Initial State: New Session

   Input: $f(V) = x + y - z == f(V) = (x + y) - z, x = [2, 4], y = [3, 5], z = [2, 2]$

   Output: True

   How test will be performed: Unit Testing

2. test-parse_commutativityOfMultiplicationDivision

   Type: Functional, Automatic, System, Regression

   Initial State: New Session

   Input: $f(V) = x * y/z == f(V) = (x * y)/z, x = [2, 4], y = [3, 5], z = [2, 2]$

   Output: True

   How test will be performed: Unit Testing

3. test-parse_precedenceOfOperators1
   Type: Functional, Automatic, System, Regression
   Initial State: New Session
   Input: $f(V) = x + y * z == x + (y * z), x = [2, 4], y = [3, 5], z = [2, 2]$
   Output: True
   How test will be performed: Unit Testing

4. test-parse_precedenceOfOperators2
   Type: Functional, Automatic, System, Regression
   Initial State: New Session
   Input: $f(V) = 2^x * y == (2^x) * y, x = [2, 4], y = [3, 5]$
   Output: True
   How test will be performed: Unit Testing

5. test-parse_precedenceOfOperators3
   Type: Functional, Automatic, System, Regression
   Initial State: New Session
   Input: $f(V) = x^2 * y == (x^2) * y, x = [2, 4], y = [3, 5]$
   Output: True
   How test will be performed: Unit Testing

6. test-parse_precedenceOfOperators4
   Type: Functional, Automatic, System, Regression
   Initial State: New Session
   Input: $f(V) = x * (y + z), x = [2, 4], y = [3, 5], z = [2, 2]$
   Output: $[10, 28]$
   How test will be performed: Unit Testing

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.

The tests that follow these assume that the user input tests (5.1.1) and input conditioning tests (5.1.2) have passed.

Correctness of Addition, Subtraction, and Multiplication

1. test-calculate_addition

   Type: Functional, Automatic, Integration

   Initial State: New Session

   Input: $f(V) = x + y$, $x = [2, 4], y = [3, 5]$

   Output: $[5, 9]$

   How test will be performed: Unit Testing

2. test-calculate_subtraction

   Type: Functional, Automatic, Integration

   Initial State: New Session

   Input: $f(V) = x - y$, $x = [2, 4], y = [3, 5]$

   Output: $[-1, -1]$

   How test will be performed: Unit Testing

3. test-calculate_multiplication1

   Type: Functional, Automatic, Integration

   Initial State: New Session

   Input: $f(V) = x * y$, $x = [2, 4], y = [3, 5]$

   Output: $[6, 20]$

   How test will be performed: Unit Testing

4. test-calculate_multiplication2
   Type: Functional, Automatic, Integration
   Initial State: New Session
   Input: $f(V) = x * y$, $x = [-1, 3], y = [-3, 5]$
   Output: $[-9, 15]$
   How test will be performed: Unit Testing

5. test-calculate_multiplication3
   Type: Functional, Automatic, Integration
   Initial State: New Session
   Input: $f(V) = x * y$, $x = [-3, -1], y = [-5, -2]$
   Output: $[2, 15]$
   How test will be performed: Unit Testing

Correctness of Division

1. test-calculate_divisionPositiveDivisor1
   Type: Functional, Automatic, Integration
   Initial State: New Session
   Input: $f(V) = x/y$, $x = [2, 4], y = [3, 5]$
   Output: [0.4, 1.33333333333333]
   How test will be performed: Unit Testing

2. test-calculate_divisionPositiveDivisor2
   Type: Functional, Automatic, Integration
   Initial State: New Session
   Input: $f(V) = x/y$, $x = [0, 4], y = [3, 5]$
   Output: [0, 1.33333333333333]
   How test will be performed: Unit Testing

3. test-calculate_divisionPositiveDivisor3
   Type: Functional, Automatic, Integration
   Initial State: New Session
   Input: $f(V) = x/y$, $x = [-1, 4], y = [3, 5]$
   Output: $[-0.33333333333, 0.8]$
   How test will be performed: Unit Testing

4. test-calculate_divisionPositiveDivisor4
   Type: Functional, Automatic, Integration
   Initial State: New Session
   Input: $f(V) = x/y$, $x = [-2, -1], y = [3, 5]$
   Output: $[-0.6666666666, -0.2]$
   How test will be performed: Unit Testing

5. test-calculate_divisionPositiveDivisor5
   Type: Functional, Automatic, Integration
   Initial State: New Session
   Input: $f(V) = x/y$, $x = [-2, 0], y = [3, 5]$
   Output: $[-0.6666666666, 0]$
   How test will be performed: Unit Testing

6. test-calculate_divisionNegativeDivisor1
   Type: Functional, Automatic, Integration
   Initial State: New Session
   Input: $f(V) = x/y$, $x = [2, 4], y = [-3, -5]$
   Output: $[-0.8, -0.6666666666]$
   How test will be performed: Unit Testing

7. test-calculate_divisionNegativeDivisor2
   Type: Functional, Automatic, Integration
   Initial State: New Session
   Input: $f(V) = x/y$, $x = [0, 4], y = [-3, -5]$
   Output: $[-0.8, 0]$
   How test will be performed: Unit Testing

8. test-calculate_divisionNegativeDivisor3
   Type: Functional, Automatic, Integration
   Initial State: New Session
   Input: $f(V) = x/y$, $x = [-1, 4], y = [-3, -5]$
   Output: $[-0.8, 0.2]$
   How test will be performed: Unit Testing

9. test-calculate_divisionNegativeDivisor4
   Type: Functional, Automatic, Integration
   Initial State: New Session
   Input: $f(V) = x/y$, $x = [-2, 0], y = [-3, -5]$
   Output: $[0, 0.4]$
   How test will be performed: Unit Testing

10. test-calculate_divisionNegativeDivisor5
    Type: Functional, Automatic, Integration
    Initial State: New Session
    Input: $f(V) = x/y$, $x = [-2, -1], y = [-3, -5]$
    Output: $[0.33333333333, 0.4]$
    How test will be performed: Unit Testing

11. test-calculate_mixedDivisorComponent

    Type: Functional, Automatic, Integration

    Initial State: New Session

    Input: $f(V) = x/(y * z)$, $x = [-2, -1], y = [3, 5], z = [-1, 1]$

    Output: Error – Cannot complete operation because a mixed interval was calculated for a divisor ->$R(f(V)) = R([-2, -1]/[-5, 5])$

    How test will be performed: Unit Testing

Correctness of Intervals as Exponents

1. test-calculate_intervalAsExponents

   Type: Functional, Automatic, Integration

   Initial State: New Session

   Input: $f(V) = 2^x$, $x = [2, 4]$

   Output: $[4, 16]$

   How test will be performed: Unit Testing

Correctness of Intervals as Base Numbers

1. test-parse_intervalWithExponent1

   Type: Functional, Automatic, Integration

   Initial State: New Session

   Input: $f(V) = x^3$, $x = [2, 4]$

   Output: $[8, 64]$

   How test will be performed: Unit Testing

2. test-parse_intervalWithExponent2

   Type: Functional, Automatic, Integration

   Initial State: New Session

Input: $f(V) = x^2$, $x = [2, 4]$

Output: $[4, 16]$

How test will be performed: Unit Testing

3. test-parse_intervalWithExponent3

   Type: Functional, Automatic, Integration

   Initial State: New Session

   Input: $f(V) = x^2$, $x = [0, 4]$

   Output: $[0, 16]$

   How test will be performed: Unit Testing

4. test-parse_intervalWithExponent4

   Type: Functional, Automatic, Integration

   Initial State: New Session

   Input: $f(V) = x^2$, $x = [-2, 4]$

   Output: $[0, 16]$

   How test will be performed: Unit Testing

5. test-parse_intervalWithExponent5

   Type: Functional, Automatic, Integration

   Initial State: New Session

   Input: $f(V) = x^2$, $x = [-4, -2]$

   Output: $[4, 16]$

   How test will be performed: Unit Testing

5.1.4 Composing Operators

This test suite is designed to determine if the R6 [Very nice to have explicit cross-references between documents. To keep the information up to date, consider using make to build your documentation. —SS] (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.

Correctness of Operator Composition

1. test-parse_multipleOperatorsInFX

   Type: Static, Manual, Unit

   Initial State: New Session

   Input: $R(f(V)) = R(f_1(V)) < op > R(f_2(V))$, where $R(f_1(V)), R(f_2(V))$ exist

   Output: True [What does True mean? $R(f)$ does not have the type Boolean? —SS]

   How test will be performed: Induction

2. test-parse_multipleOperatorsInFX

   Type: Static, Manual, Unit

   Initial State: New Session

   Input: $R(f(V)) = R(f_1(V)) < op > R(f_2(V))$, where $R(f_1(V))$ or $R(f_2(V))$ do not exist

   Output: Error – Calculation path encountered an unsupported interval operation

   How test will be performed: Induction

5.1.5 Presentation of Calculations to the User

This test suite is designed to test that the final functional requirements, R9 (Showing the results to the user or an error if a result cannot be found) and R10 (Representing $R(f(V))$ with the most precise number of significant decimal figures), are 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.

The tests that follow these assume that the user input tests (5.1.1), input conditioning tests (5.1.2), component equation calculation tests (5.1.3), and operator composition tests (5.1.4) have passed.

Rounding For Significant Figures

1. test-equalSignificantFigures

   Type: Functional, Automatic, System, Regression

   Initial State: New Session

   Input: $f(V) = x + y, x = [2.5, 4.3], y = [3.5, 5.5]$

   Output: $[6.0, 9.8]$

   How test will be performed: Unit Testing

2. test-unequalSignificantFigures

   Type: Functional, Automatic, System, Regression

   Initial State: New Session

   Input: $f(V) = x + y, x = [2.5, 4.33], y = [3.45, 5.555]$

   Output: $[6.0, 9.9]$

   How test will be performed: Unit Testing

3. test-significantFiguresFloatingPoint

   Type: Functional, Automatic, System, Regression

   Initial State: New Session

   Input: $f(V) = x+y, x = [2.5343...3, 4.3333..34], y = [3.5343...3, 5.5343...3]$

Output: [6.0686...6, 9.8676...6]
Warning – Precision of domain boundaries exceeds the floating point representation on this machine ->rounded to the most significant figure available

How test will be performed: Unit Testing

Note: Numerical values for input and output are for illustrative purposes only

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), robustness (applicable at every process stage), and reliability (5.1.5). 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 covered.

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.

[Great to see usability testing! Could you please give some more detail on how the tests will be run. How many participants? How will they be recruited? What documentation will they be given? etc. What specific task or tasks will they be asked to perform? How will the results of the usability tests be summarized? —SS]

Use of Standard Mathematical Notation

1. test-enteringFV

   Type: Non-Functional, Manual, System

   Initial State: New Session

   Condition: The user is given a series of $f(V)$ to enter into the tool; the $f(V)$ chosen will be taken from the functional requirement tests (5.1)

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

How test will be performed: Usability questionnaire (7.1)

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: Usability questionnaire (7.1)

GUI Presentation and Organization

1. test-GUIPresentationInputs

   Type: Non-Functional, Manual, System

   Initial State: New Session

   Condition: The user will be given an input set $\{f(V), D(v)v \in V\}$ and asked to enter them into the GUI

   Result: The user should be able to find and enter the data into the GUI with no guidance

   How test will be performed: Usability questionnaire (7.2)

2. test-GUIWorkflow

   Type: Non-Functional, Manual, System

   Initial State: New Session

   Condition: The user will be shown a GUI that has an input set $\{f(V), D(v)v \in V\}$ already entered and ask them to tell the tool to compute the result

   Result: The user should be able to prompt the tool to begin processing with no guidance

How test will be performed: Usability questionnaire (7.2)

3. test-GUIPresentationOutputs

   Type: Non-Functional, Manual, System

   Initial State: End Session

   Condition: The user will be shown a GUI that has finished 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: Usability questionnaire (7.2)

5.3 Traceability Between Test Cases and Requirements

The purpose of the traceability matrix (Table 1) and graph (Figure **??**) [Label and reference do not match —SS] is to provide easy references between the test suite and the requirements from the SRS (version 1.1.1). 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	R10	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	X	
5.2.1											X

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

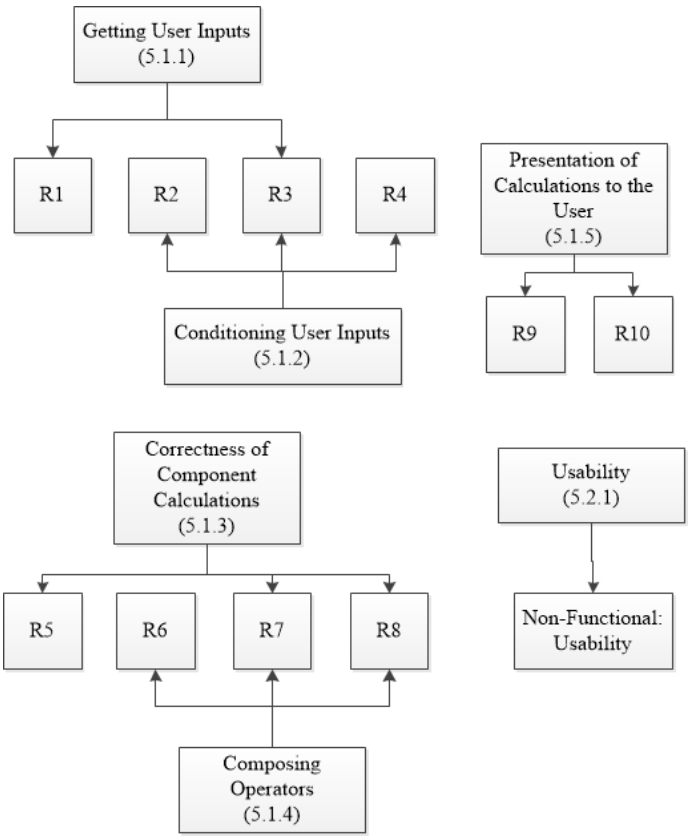


Figure 1: Traceability Graph Showing the Connections Requirements and Test Suites

6 Unit Testing Plan

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. This also has implications for specific function unit tests.

The general plan for unit testing is to use the same tests from the functional requirements (Section 5.1) except that the expected inputs are provided by the human tester rather than the module that produces those values as outputs. For example if the $f(V)$ parsing function is expecting an error-checked $f(V)$ from the input module, the error-checked $f(V)$ would be provided by the human tester instead. If a manually synthesised input produces incorrect behaviours from a

function, debugging strategies can be employed, including debugging statements and visual code inspections.

The unit testing stage is also the point where code coverage tools can be used to ensure that most lines of code are being tested by the integration and system tests. If code is not covered by the tests, it should be analysed to determine its usefulness and accessibility to the rest of the program. [You should explicitly state that your goal is 100% code coverage. —SS]

Once a section of tests have been completed and there do not appear to be any immediate problems, the modules can be used in the functional requirement tests.

7 Appendix: Usability Survey Questions

These surveys are designed to accompany the usability non-functional requirements tests (5.2.1).

7.1 User Questions for Entering Values into the C^3 Tool

1. (Likert Scale) I was able to enter in the equation $f(V)$ in a format that I was familiar with.
2. (Likert Scale) I was able to enter in the equation $f(V)$ in a format that I was expecting.
3. (Likert Scale) I found it easy to enter the $f(V)$ equations into the tool.
4. (Likert Scale) I was able to enter in the domains $D(v)$ in a format that I was familiar with.
5. (Likert Scale) I was able to enter in the domains $D(v)$ in a format that I was expecting.
6. (Likert Scale) I found it easy to enter the domains $D(v)$ into the tool.
7. (Open Response) Do you have any suggestions for improving the mathematical notation of the C^3 tool?

7.2 User Questions for Using the C^3 Tool GUI

1. (Likert Scale) I found it easy to find the input mechanism for entering $f(V)$ into the GUI.
2. (Likert Scale) I found it easy to enter the $f(V)$ equations into the GUI.
3. (Likert Scale) I found it easy to find the input mechanism for entering $D(v)$ into the GUI.
4. (Likert Scale) I found it easy to enter the domains $D(v)$ into the GUI.
5. (Likert Scale) I found it easy to find the program's result for $R(f(V))$ in the GUI.

6. (Likert Scale) I found it easy to understand the program's result for $R(f(V))$ in the GUI.

7. (Likert Scale) I found it easy to understand the sequence of steps required to use the C^3 tool.

8. (Open Response) Do you have any suggestions for improving the process flow of the C^3 tool?