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

Geneva Smith

October 16, 2017

1 Revision History

Date	Version	Notes
	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

[symbols, abbreviations or acronyms – you can reference the SRS tables if needed —SS]

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, $R(f(X))$, of a user-specified function, $f(X)$, given the domains of the function's variables, $D(X)$. 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 1.0.1), 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 1.0.1). Any functionality defined beyond the scope of the SRS document are not included in this version of the test plan.

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 sequence of continuous 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 mathematics.

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 1.0.1). 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.

4.4 Verification Tools

A programming language has not been selected for this project yet, but the options have been limited to Java and C# because of their GUI building capabilities. Each language has an IDE with a number of built-in or plug-in based test tools. Even though a number of tools are mentioned for each language, they might not all be used in the verification process of the C^3 tool.

The Eclipse Oxygen IDE for Java Developers would be used if Java is selected as the implementation language. This includes unit testing with JUnit4, code coverage with JaCoCo, and performance profiling. There is also the potential for coded GUI testing with Eclipse plugins such as WindowTester and Squish.

The Visual Studio 2017 (Enterprise Edition) IDE, which would be used if C# is chosen as the implementation language, also comes with a number of test tools which could be utilised during the C^3 tool verification. These tools include a unit testing framework and management system, coded GUI testing, and code coverage tools. Unit testing in this IDE can also be performed "live" such that code is checked for test coverage and conformity as code is written.

4.5 Non-Testing Based Verification

There are no non-testing based verification approaches in this test plan.

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 $f(X)$ 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.

5.1 Tests for Functional Requirements

5.1.1 Getting User Inputs

This test suite is designed to determine if the **??** functional requirement is satisfied.

User Inputs (As Expected)

1. test-directinput

   Type: Functional

   Initial State: New Session

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

   Output: Input received from direct input

   How test will be performed: Unit Test

2. test-fileinput

   Type: Functional

   Initial State: New Session

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

Output: Input received from file

How test will be performed: Unit Test

3. test-badFileInput

   Type: Functional

   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

   Initial State: New Session

   Input: $f(X) = 34$

   Output: N/A

   How test will be performed: Unit Test

User Inputs (Extraneous Information)

1. test-input_variableNotInFunction

   Type: Functional

   Initial State: New Session

   Input: $f(X) = 34$, $x = [2, 4]$

   Output: Warning – Function $f(X)$ does not contain name x found in variable list

   How test will be performed: Unit Test

User Inputs (Missing Values)

1. test-input_noFunction
   Type: Functional
   Initial State: New Session
   Input: $x = [2, 4], y = [2, 4]$
   Output: Error – Could not find function $f(X)$ with variables x, y
   How test will be performed: Unit Test

User Inputs (Incomplete Values)

1. test-input_missingFunctionValue
   Type: Functional
   Initial State: New Session
   Input: $f(X) = x+$, $x = [2, 4], y = [3, 5]$
   Output: Error – Function $f(X)$ is missing an operand for +
   How test will be performed: Unit Test

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

3. test-input_missingMaxDomainValue
   Type: Functional
   Initial State: New Session
   Input: $\{f(X) = x + y, x = [2, 4], y = [3,]\}$, $\{f(X) = x + y, x = [2, 4], y = [3]\}$
   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

   Initial State: New Session

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

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

   How test will be performed: Unit Test

5.1.2 Conditioning User Inputs

This test suite is designed to determine if the **??** and **??** functional requirements are satisfied.

Parsing User Inputs (Single Operators)

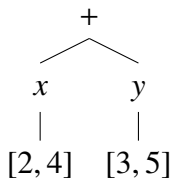
1. test-parse_addition

   Type: Functional

   Initial State: New Session

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

   Output:

   How test will be performed: Unit Testing

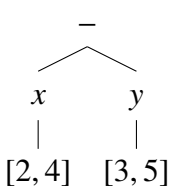
2. test-parse_subtraction

   Type: Functional

   Initial State: New Session

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

Output:



How test will be performed: Unit Testing

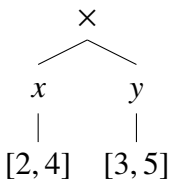
3. test-parse_multiplication

   Type: Functional

   Initial State: New Session

   Input: $f(X) = x \times y$, $x = [2, 4], y = [3, 5]$

   Output:



   How test will be performed: Unit Testing

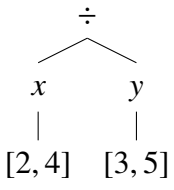
4. test-parse_division

   Type: Functional

   Initial State: New Session

   Input: $f(X) = x \div y$, $x = [2, 4], y = [3, 5]$

   Output:



   How test will be performed: Unit Testing

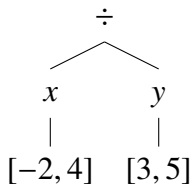
5. test-parse_divisionMixedIntervalQuotient

   Type: Functional

Initial State: New Session

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

Output:



How test will be performed: Unit Testing

6. test-parse_divisionMixedIntervalDivisor

   Type: Functional

   Initial State: New Session

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

   Output: Error – Cannot perform division with a mixed interval divisor

   How test will be performed: Unit Testing

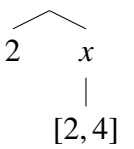
7. test-parse_intervalAsExponents

   Type: Functional

   Initial State: New Session

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

   Output:



   How test will be performed: Unit Testing

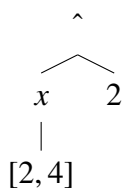
8. test-parse_intervalWithExponent

   Type: Functional

   Initial State: New Session

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

Output:



How test will be performed: Unit Testing

9. test-parse_intervalsOnlyExponentiation

   Type: Functional

   Initial State: New Session

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

   Output: Error – Cannot perform exponentiation when the base number and the exponent are intervals

   How test will be performed: Unit Testing

10. test-parse_multipleOperatorsInFX

    Type: Static

    Initial State: New Session

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

    Output: True

    How test will be performed: Induction

11. test-parse_multipleOperatorsInFX

    Type: Static

    Initial State: New Session

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

    Output: Error – Calculation path encountered an unsupported interval operation

How test will be performed: Induction

5.1.3 Verifying Data Constraints

This test suite is designed to determine if the **??** functional requirement is satisfied.

Domain Constraints

1. 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

Exponentiation Constraints

1. test-parse_intervalAsExponentsInvalidBase

   Type: Functional

   Initial State: New Session

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

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

   How test will be performed: Unit Testing

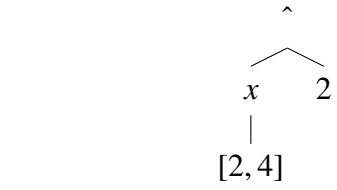
2. test-parse_intervalWithInvalidExponent1

   Type: Functional

   Initial State: New Session

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

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



$[2, 4]$

How test will be performed: Unit Testing

3. test-parse_intervalWithInvalidExponent2

   Type: Functional

   Initial State: New Session

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

   Output: Error – Cannot perform operation exponentiation with an exponent < 0

   How test will be performed: Unit Testing

5.2 Tests for Non-Functional Requirements

5.2.1 Area of Testing1

Title for Test

1. test-id1

   Type:

   Initial State:

   Input/Condition:

   Output/Result:

   How test will be performed:

2. test-id2

Type: Functional, Dynamic, Manual, Static etc.

Initial State:

Input:

Output:

How test will be performed:

5.2.2 Area of Testing2

...

5.3 Traceability Between Test Cases and Requirements

6 Unit Testing Plan

[Unit testing plans for internal functions and, if appropriate, output files —SS]

References

7 Appendix

This is where you can place additional information.

7.1 Symbolic Parameters

The definition of the test cases will call for SYMBOLIC_CONSTANTS. Their values are defined in this section for easy maintenance.

7.2 Usability Survey Questions?

This is a section that would be appropriate for some teams.