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

Geneva Smith

October 12, 2017

1 Revision History

Date	Version	Notes
	1.0	Initial draft completed

2 Symbols, Abbreviations and Acronyms

symbol	description
C^3	Companion Cube Calculator
SRS	Software Requirements Specification
T	Test

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

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 . . . 1

4 Plan **1**
- 4.1 Software Description . . . 1
- 4.2 Test Team . . . 1
- 4.3 Automated Testing Approach . . . 2
- 4.4 Verification Tools . . . 2
- 4.5 Non-Testing Based Verification . . . 2

5 System Test Description **2**
- 5.1 Tests for Functional Requirements . . . 2
  - 5.1.1 Getting User Inputs . . . 2
  - 5.1.2 Conditioning User Inputs . . . 5
- 5.2 Tests for Non-Functional Requirements . . . 6
  - 5.2.1 Area of Testing1 . . . 6
  - 5.2.2 Area of Testing2 . . . 7
- 5.3 Traceability Between Test Cases and Requirements . . . 7

6 Unit Testing Plan **7**

7 Appendix **8**
- 7.1 Symbolic Parameters . . . 8
- 7.2 Usability Survey Questions? . . . 8

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, $D(X)$.

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), 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). Any functionality defined beyond the scope of the SRS document are not included in this version of the test plan.

3.2 Scope

3.3 Overview of Document

4 Plan

4.1 Software Description

4.2 Test Team

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

4.3 Automated Testing Approach

4.4 Verification Tools

[Thoughts on what tools to use, such as the following: unit testing framework, valgrind, static analyzer, make, continuous integration, test coverage tool, etc. —SS]

4.5 Non-Testing Based Verification

[List any approaches like code inspection, code walkthrough, symbolic execution etc. Enter not applicable if that is the case. —SS]

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

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_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

5.1.2 Conditioning User Inputs

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

User Inputs (Single Operators)

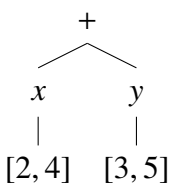
1. test-directinput_addition

   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

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.