Test Report: Companion Cube Calculator

Geneva Smith

December 19, 2017

1 Revision History

Date	Version	Notes
December 19, 2017 1	1.0	Completed the initial test report

2 Symbols, Abbreviations and Acronyms

symbol	description
R	Requirement
T	Test

Contents

List of Tables

3 Introduction

This is the test report for the Companion Cube Calculator, a mathematical tool which determines the range of a user-specified function given the domains of the function's variables. The the directory for this project can be found at:

https://github.com/GenevaS/CAS741.

4 Functional Requirements Evaluation

The following tests have failed by verification:

Table 1: Failed Functional Test Summary

ID	Input	Expected Outcome	Expected MsgID	Actual MsgID
test-control_ precedenceOf-Operators3	`"x^2*y", "x,2,4\ny,3,5", "(x^2)*y", "x,2,4\ny,3,5"`	*TRUE*	-	(EQC_ INCOMPLETE_OP) Error: Unrecognized sequence encountered during Atomic Equation parsing. Remaining equation = `)*y`.
test-control_ precedenceOf-Operators6	`"(2(x+y)^2) /(3^z)", "x,1,2\ny,3,4 \nz,5,6"`	`"[0.0438 9574759945 13, \n 0.2962962 96296296]"`	Range calculated successfully.	(EQC_ INCOMPLETE_OP) Error: Unrecognized sequence encountered during Atomic Equation parsing. Remaining equation = `)/(3^z)`.

5 Non-Functional Requirements Evaluation

The non-functional requirements were not always easy to verify. Some, such as robustness, could be adequately verified using the existing test suites. Others, such as correctness and usability, required manual testing. Not all non-functional requirements were verified in this project cycle due to time constraints.

5.1 Correctness

Correctness testing could not be completed in this project cycle due to time constraints.

These tests are directly related to R4 and R6 (Decomposing the user equation into components and recomposing the results).

5.2 Robustness

The robustness requirement for recognizing violated data constraints was covered in the functional tests:

- The constraints on supported operators are contained in the Range Solver test suite. The containment of all operator-specific information within this module made it possible to collect all of these restrictions in the same suite. This design also implicitly supported the output constraint on $R(f(V))$ because only mathematical operations that produced closed, real intervals were implemented.
- The constraint of having every $D(v) \in V$ defined as a closed, real interval are contained in the Interval Conversion and Interval Data Structure modules.

These tests are directly related to R3 and R8 (Verifying that the program satisfies the input and output constraints).

5.3 Verifiability

The verifiability requirement stated that the program must be created in a way in which its calculations can be checked for correctness. By basing this design on verifiable mathematical concepts and implementing the equation decomposition using a grammar definition, it is possible to measure if this requirement has met. However, verifiability testing could not be completed in this project cycle due to time constraints.

This is indirectly related to R9 because the outputs must be shown to the program user such that they understand and have confidence in the program's results.

5.4 Usability

5.5 Maintainability

The maintainability requirements focus on the extensibility of the original implementation with respect to its supported mathematical operations. Support for open, real intervals already exists in the Interval Conversion and Interval Data Structure modules. This means that it is possible that only the Range Solver module would need to be updated to add more mathematical operations. However, maintainability testing could not be completed in this project cycle due to time constraints.

6 Unit Testing

In addition to the functional requirements, unit tests were implemented to achieve 100% code coverage in all documented modules. The purpose of this was to ensure that all code paths, including conditional branches, were being executed and to help identify program errors that were not covered in the functional testing suite. Implementation files that were automatically generated or that were added to implement the GUI are not covered in the unit tests.

Table 2: Unit Test Summary

Test Suite	**Test File**	**Target Modules**	**Total Tests**	**Tests Passing (%)**
Control Flow	`ControlTests.cs`	ControlFlow (MIS: 6)	6	100%
User Input	`InputTests.cs`	Input (MIS: 7)	10	100%
Interval	`IntervalTests.cs`	Interval Data Structure (MIS: 13), Interval Conversion (MIS: 8)	7	100%
Equation	`EquationTests.cs`	Equation Data Structure (MIS: 14), Equation Conversion (MIS: 9)	16	100%
Variable Consolidation	`VariableConsolidationTests.cs`	Consolidate (MIS: 10)	8	100%
Solver	`SolverTests.cs`	Operator Data Structure (MIS: 15), Solver (MIS: 11)	17	100%
Output	`OutputTests.cs`	Output (MIS: 12)	6	100%

7 Changes Due to Testing

The test suites uncovered a number of bugs, most of which resulted in changes to the equation parsing process in the Equation Conversion module (MIS: 9). The failed functional tests have identified an additional parsing error that will be corrected in a future project cycle.

The user study also uncovered a number of failings, including the inability to copy the range values, which will also be addressed in a future project cycle.

8 Automated Testing

All automated testing was performed using the unit testing framework available in Visual Studio 2017 (Enterprise Edition). The associated project is `UnitTests_CompanionCubeCalculator.csproj`.

9 Trace to Requirements

									R1
									R2
									R3
									R4
									R5
									R6
									R7
									R8
									R9
									NF: Robustness
									NF: Usability

Table 3: Traceability Matrix Showing the Connections Between Requirements and Test Suites

10 Trace to Modules

	Control Flow	User Input	Interval	Equation	Variable Consolidation	Solver	Output
MIS: 6	X						
MIS: 7	X	X					
MIS: 8			X		X		
MIS: 9				X	X		
MIS: 10	X				X		
MIS: 11	X					X	
MIS: 12	X						X
MIS: 13	X		X		X	X	X
MIS: 14	X			X	X	X	X
MIS: 15				X	X	X	

Table 4: Traceability Matrix Showing the Connections Between Modules and Test Suites

11 Code Coverage Metrics

A coverage of 100% has been achieved for all documented modules. Code coverage was not considered for additional files that were created to support the GUI.