Module Guide: Companion Cube Calculator (C^3)

Geneva Smith

December 10, 2017

1 Revision History

Date	Version	Notes
December 10, 2017	1.1.1	Added justification for assigning multiple anticipated changes to one module.
December 5, 2017	1.1	Added a description for an operator data structure module
November 2, 2017	1.0	Completed first document draft

Contents

List of Tables

List of Figures

2 Introduction

Decomposing a system into modules is a commonly accepted approach to developing software. A module is a work assignment for a programmer or programming team (Parnas et al., 1984). We advocate a decomposition based on the principle of information hiding (Parnas, 1972). This principle supports design for change, because the "secrets" that each module hides represent likely future changes. Design for change is valuable in scientific computing, where modifications are frequent, especially during initial development as the solution space is explored.

Our design follows the rules proposed by Parnas et al. (1984), as follows:

- System details that are likely to change independently should be the secrets of separate modules.
- Each data structure is used in only one module.
- Any other program that requires information stored in a module's data structures must obtain it by calling access programs belonging to that module.

After completing the first stage of the design, the Software Requirements Specification (SRS), the Module Guide (MG) is developed (Parnas et al., 1984). The MG specifies the modular structure of the system and is intended to allow both designers and maintainers to easily identify the parts of the software. The potential readers of this document are:

- New project members: This document can be a guide for a new project member to easily understand the overall structure and quickly find the relevant modules they are searching for.
- Maintainers: The hierarchical structure of the module guide improves the maintainers' understanding when they need to make changes to the system. It is important for a maintainer to update the relevant sections of the document after changes have been made.
- Designers: Once the module guide has been written, it can be used to check for consistency, feasibility and flexibility. Designers can verify the system in various ways, such as consistency among modules, feasibility of the decomposition, and flexibility of the design.

The rest of the document is organized as follows. Section 3 lists the anticipated and unlikely changes of the software requirements. Section 4 summarizes the module decomposition that was constructed according to the likely changes. Section 5 specifies the connections between the software requirements and the modules. Section 6 gives a detailed description of the modules. Section 7 includes two traceability matrices. One checks the completeness of the design against the requirements provided in the SRS. The other shows the relation between anticipated changes and the modules. Section 8 describes the use relation between modules.

3 Anticipated and Unlikely Changes

This section lists possible changes to the system. According to the likeliness of the change, the possible changes are classified into two categories. Anticipated changes are listed in Section 3.1, and unlikely changes are listed in Section 3.2.

3.1 Anticipated Changes

Anticipated changes are the source of the information that is to be hidden inside the modules. Ideally, changing one of the anticipated changes will only require changing the one module that hides the associated decision. The approach adapted here is called design for change.

AC1: The specific hardware on which the software is running.

AC2: The format of the input data.

AC3: The format of the output data.

AC4: The constraints on the output data.

AC5: The implementation of the equation data structure.

AC6: The constraints on the allowable user equation operators.

AC7: The implementation of the operator data structure.

AC8: The implementation of the interval data structure.

AC9: The constraints on intervals.

AC10: The algorithm for decomposing the user equation.

AC11: The algorithm used to calculate the range of the user equation.

AC12: The output of a graphical version of the equation parse tree.

3.2 Unlikely Changes

The module design should be as general as possible. However, a general system is more complex. Sometimes this complexity is not necessary. Fixing some design decisions at the system architecture stage can simplify the software design. If these decision should later need to be changed, then many parts of the design will potentially need to be modified. Hence, it is not intended that these decisions will be changed.

UC1: Input/Output devices (Input: File and/or Keyboard, Output: File, Memory, and/or Screen).

UC2: The purpose of the C^3 tool is always to calculate the range of an equation given the domains of its input variables.

UC3: The decomposition of the user equation will always follow BEDMAS rules.

4 Module Hierarchy

This section provides an overview of the module design. Modules are summarized in a hierarchy decomposed by secrets in Table 1. The modules listed below, which are leaves in the hierarchy tree, are the modules that will actually be implemented.

M1: Hardware-Hiding Module

M2: Control Flow Module

M3: User Input Module

M4: Interval Conversion Module

M5: Equation Conversion Module

M6: Variable Consolidation Module

M7: Range Solver Module

M8: Output Module

M9: Interval Data Structure Module

M10: Equation Data Structure Module

M11: Operator Data Structure Module

Level 1	**Level 2**
Hardware-Hiding Module	-
Behaviour-Hiding Module	Control Flow Module User Input Module Interval Conversion Module Equation Conversion Module Variable Consolidation Module Range Solver Module Output Module
Software Decision Module	Interval Data Structure Module Equation Data Structure Module Operator Data Structure Module

Table 1: Module Hierarchy

5 Connection Between Requirements and Design

The design of the system is intended to satisfy the requirements developed in the SRS. In this stage, the system is decomposed into modules. The connection between requirements and modules is listed in Table 2.

6 Module Decomposition

Modules are decomposed according to the principle of "information hiding" proposed by Parnas et al. (1984). The *Secrets* field in a module decomposition is a brief statement of the design decision hidden by the module. The *Services* field specifies *what* the module will do without documenting *how* to do it. For each module, a suggestion for the implementing software is given under the *Implemented By* title. If the entry is *OS*, this means that the module is provided by the operating system or by standard programming language libraries. Also indicate if the module will be implemented specifically for the software.

Only the leaf modules in the hierarchy have to be implemented. If a dash (–) is shown, this means that the module is not a leaf and will not have to be implemented. Whether or not this module is implemented depends on the programming language selected.

6.1 Hardware Hiding Modules (M1)

Secrets: The data structure and algorithm used to implement the virtual hardware.

Services: Serves as a virtual hardware used by the rest of the system. This module provides the interface between the hardware and the software. So, the system can use it to display outputs or to accept inputs.

Implemented By: OS

6.2 Behaviour-Hiding Module

Secrets: The contents of the required behaviours.

Services: Includes programs that provide externally visible behaviour of the system as specified in the software requirements specification (SRS) documents. This module serves as a communication layer between the hardware-hiding module and the software decision module. The programs in this module will need to change if there are changes in the SRS.

Implemented By: Companion Cube Calculator

6.2.1 Control Flow Module (M2)

Secrets: The calling order of program modules.

Services: Facilitates and coordinates the flow of information and removes direct dependencies between modules.

Implemented By: Companion Cube Calculator

6.2.2 User Input Module (M3)

Secrets: The method of acquiring user inputs (console input, GUI, file).

Services: Gathers data from the user input mechanism and converts it into a standard intermediary format.

Implemented By: Companion Cube Calculator

6.2.3 Interval Conversion Module (M4)

Secrets: The algorithm for parsing variable domain values into intervals.

Services: Converts the user-defined list of variable domains into intervals. This module checks for syntactic errors in the data set (e.g. missing boundary values). This module produces warnings for data that is valid, but might not have been intended (e.g. minimum and maximum values are switched, boundaries that have the same value).

Implemented By: Companion Cube Calculator

6.2.4 Equation Conversion Module (M5)

Secrets: The algorithm for parsing the user equation.

Services: Decomposes the user equation into a series of two-operator equations and checks for syntactic errors (e.g. open brackets, too few operands, missing or extraneous variables). This module produces warnings for equations that are valid, but might not have been intended (e.g. $f(V) = 34$).

Implemented By: Companion Cube Calculator

6.2.5 Variable Consolidation Module (M6)

Secrets: The method for checking the compatibility of the user-provided equation and the user-provided variable list.

Services: Facilitates the user data parsing process and ensures that the resulting equation and interval list are compatible with respect to the variable names used in the equation.

Implemented By: Companion Cube Calculator

6.2.6 Range Solver Module (M7)

Secrets: The algorithm for solving the user equation and constraints on the supported operators and interval boundaries.

Services: Defines what operators are supported and the associated constraints. Calculates the range of the user equation using interval data.

Implemented By: Companion Cube Calculator

6.2.7 Output Module (M8)

Secrets: The output format and the output interface.

Services: Converts the internal data representation into a human readable format and passes the information to the target output device.

Implemented By: Companion Cube Calculator

6.3 Software Decision Module

Secrets: The design decision based on mathematical theorems, physical facts, or programming considerations. The secrets of this module are *not* described in the SRS.

Services: Includes data structure and algorithms used in the system that do not provide direct interaction with the user.

Implemented By: -

6.3.1 Interval Data Structure Module (M9)

Secrets: The internal representation of intervals.

Services: Stores the components of an interval and provides mechanisms for reading them.

Implemented By: Companion Cube Calculator

6.3.2 Equation Data Structure Module (M10)

Secrets: The internal representation of the user equation.

Services: Stores the structure of the equation and provides mechanisms for reading the structure.

Implemented By: Companion Cube Calculator

6.3.3 Operator Data Structure Module (M11)

Secrets: The internal representation of mathematical operators and their properties.

Services: Stores mathematical operators and properties such as precedence and associativity.

Implemented By: Companion Cube Calculator

7 Traceability Matrix

This section shows two traceability matrices: between the modules and the requirements and between the modules and the anticipated changes.

Req.	**Modules**
R1	M1, M2, M3
R2	M4, M9
R3	M2, M4, M5, M6
R4	M2, M5, M6, M7
R5	M2, M7
R6	M2, M7
R7	M7
R8	M7
R9	M1, M2, M8
R10	M8

Table 2: Trace Between Requirements and Modules

AC	**Modules**
AC1	M1
AC2	M3
AC3	M8
AC4	M8
AC5	M10
AC6	M7
AC7	M11
AC8	M9
AC9	M7
AC10	M5
AC11	M7
AC12	M8

Table 3: Trace Between Anticipated Changes and Modules

Generally, each anticipated change should map to one module because it implies that each module contains information that only it knows about the design. However, in C^3 design, two modules have multiple associated anticipated changes: the Range Solver Module (M7) and the Output Module (M8).

The Range Solver Module is responsible for the calculation of an equation's range. Since its main responsibility is to execute the algorithm used to perform the range calculation (AC11), the Range Solver Module also implicitly stores the constraints on valid operations (AC6) and intervals (AC9). Any changes to these constraints would directly result in changes to this module.

The Output Module is responsible for presenting program information to the user and all of its anticipated changes are directly related to this task. The changes include format (AC3), data constraints (AC4), and changes to the process for presenting a graphical representation of the equation tree (AC12).

8 Use Hierarchy Between Modules

In this section, the uses hierarchy between modules is provided. Parnas (1978) said of two programs A and B that A *uses* B if correct execution of B may be necessary for A to complete the task described in its specification. That is, A *uses* B if there exist situations in which the correct functioning of A depends upon the availability of a correct implementation of B. Figure 1 illustrates the use relation between the modules. It can be seen that the graph is a directed acyclic graph (DAG). Each level of the hierarchy offers a testable and usable subset of the system, and modules in the higher level of the hierarchy are essentially simpler because they use modules from the lower levels.

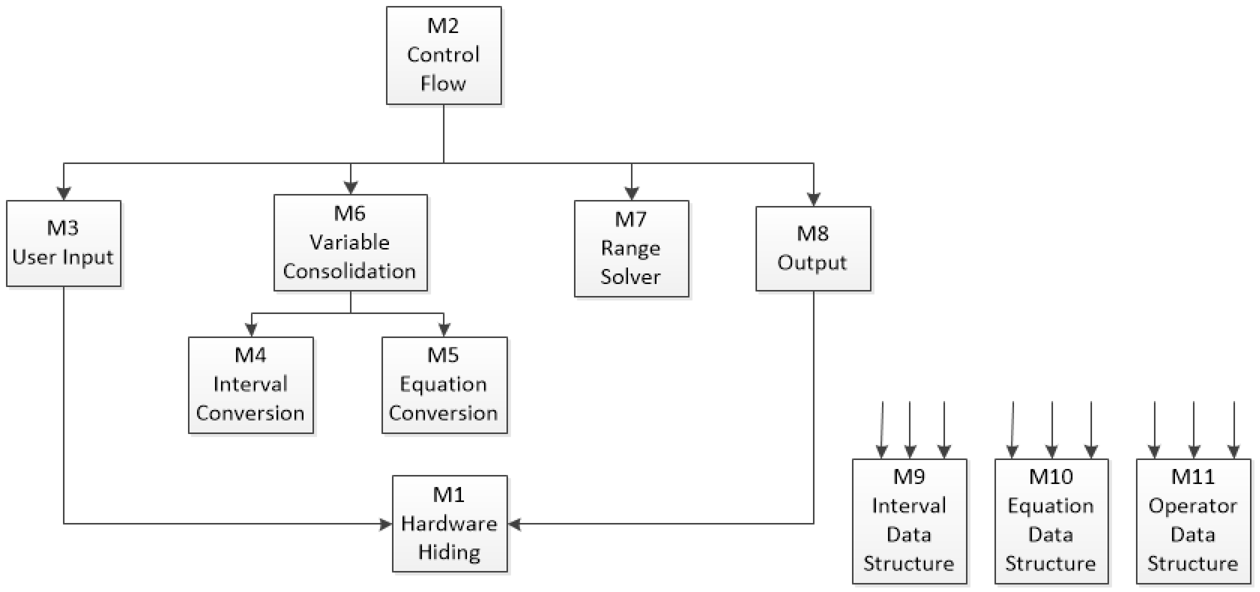

Figure 1: Use hierarchy among modules

References

David L. Parnas. On the criteria to be used in decomposing systems into modules. *Comm. ACM*, 15(2):1053–1058, December 1972.

David L. Parnas. Designing software for ease of extension and contraction. In *ICSE '78: Proceedings of the 3rd international conference on Software engineering*, pages 264–277, Piscataway, NJ, USA, 1978. IEEE Press. ISBN none.

D.L. Parnas, P.C. Clement, and D. M. Weiss. The modular structure of complex systems. In *International Conference on Software Engineering*, pages 408–419, 1984.