Module Guide for Solar Water Heating Systems Incorporating Phase Change Material

Thulasi Jegatheesan and Brooks MacLachlan

August 12, 2016

Contents

1 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). In the best practices for scientific computing (SC), Wilson et al. (2013) advise a modular design, but are silent on the criteria to use to decompose the software into modules. 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 SC, where modifications are frequent, especially during initial development as the solution space is explored.

Our design follows the rules layed out 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 as follows:

- 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 2 lists the anticipated and unlikely changes of the software requirements. Section 3 summarizes the module decomposition that was constructed according to the likely changes. Section 4 specifies the connections between the software requirements and the modules. Section 5 gives a detailed description of the modules. Section 6 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 7 describes the use relation between modules.

2 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 2.1, and unlikely changes are listed in Section 2.2.

2.1 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 initial input data.

AC3: The format of the input parameters.

AC4: The constraints on the input parameters.

AC5: The format of the final output data.

AC6: The constraints on the output results.

AC7: How the governing ODEs are defined using the input parameters.

AC8: How the energy equations are defined using the input parameters.

AC9: How the overall control of the calculations is orchestrated.

AC10: The implementation for the sequence (array) data structure.

AC11: The algorithm used for the ODE solver.

AC12: The implementation of plotting data.

2.2 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. As an example, the ODEs for the temperature and the energy equations are assumed to follow the structure given in the SRS; that is, even if they need to be modified, the modifications should be possible by changing how the input parameters are used in the definition. If new parameters are needed, this will mean a change to both the input parameters module, the calculation module and the output module.

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

UC2: There will always be a source of input data external to the software.

UC3: Output data are displayed to the output device.

UC4: The goal of the system is to calculate temperatures and energies.

UC5: The ODEs for temperature can be defined using parameters defined in the input parameters module.

UC6: The energy equations can be defined using the parameters defined in the input parameters module.

3 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: Input Format Module

M3: Input Parameters Module

M4: Input Verification Module

M5: Output Format Module

M6: Output Verification Module

M7: Temperature ODEs Module

M8: Energy Equations Module

M9: Control Module

M10: Sequence Data Structure Module

M11: ODE Solver Module

M12: Plotting Module

Note that M1 is a commonly used module and is already implemented by the operating system. It will not be reimplemented. Similarly, M10, M11 and M12 are already available in Matlab and will not be reimplemented.

Level 1	**Level 2**
Hardware-Hiding Module	
Behaviour-Hiding Module	Input Format Module Input Parameters Module Input Verification Module Output Format Module Output Verification Module Temperature ODEs Module Energy Equations Module Control Module
Software Decision Module	Sequence Data Structure Module ODE Solver Module Plotting Module

Table 1: Module Hierarchy

4 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.

5 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. If the entry is *Matlab*, this means that the module is provided by Matlab. *SWHS* means the module will be implemented by the SWHS 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.

5.1 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

5.2 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: –

5.2.1 Input Format Module (M2)

Secrets: The format and structure of the input data.

Services: Converts the input data into the data structure used by the input parameters module.

Implemented By: SWHS

5.2.2 Input Parameters Module (M3)

Secrets: The format and structure of the input parameters.

Services: Stores the parameters needed for the program, including material properties, processing conditions, and numerical parameters. The values can be read as needed. This module knows how many parameters it stores.

Implemented By: SWHS

5.2.3 Input Verification Module (M4)

Secrets: The format and structure of the physical and software constraints.

Services: Verifies that the input parameters comply with physical and software constraints. Throws an error if a parameter violates a physical constraint. Throws a warning if a parameter violates a software constraint.

Implemented By: SWHS

5.2.4 Output Format Module (M5)

Secrets: The format and structure of the output data.

Services: Outputs the results of the calculations, including the input parameters, temperatures, energies, and times when melting starts and stops.

Implemented By: SWHS

5.2.5 Output Verification Module (M6)

Secrets: The algorithm used to approximate expected results.

Services: Verifies that the output energy results follow the law of conservation of energy. Throws a warning if the relative error exceeds the error threshold.

Implemented By: SWHS

5.2.6 Temperature ODEs Module (M7)

Secrets: The ODEs for solving the temperature, using the input parameters.

Services: Defines the ODEs using the parameters in the input parameters module.

Implemented By: SWHS

5.2.7 Energy Equations Module (M8)

Secrets: The equations for solving for the energies using the input parameters.

Services: Defines the energy equations using the parameters in the input parameters module.

Implemented By: SWHS

5.2.8 Control Module (M9)

Secrets: The algorithm for coordinating the running of the program.

Services: Provides the main program.

Implemented By: SWHS

5.3 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: –

5.3.1 Sequence Data Structure Module (M10)

Secrets: The data structure for a sequence data type.

Services: Provides array manipulation, including building an array, accessing a specific entry, slicing an array, etc.

Implemented By: Matlab

5.3.2 ODE Solver Module (M11)

Secrets: The algorithm to solve a system of first order ODEs.

Services: Provides solvers that take the governing equation, initial conditions, and numerical parameters, and solve them.

Implemented By: Matlab

5.3.3 Plotting Module (M12)

Secrets: The data structures and algorithms for plotting data graphically.

Services: Provides a plot function.

Implemented By: Matlab

6 Traceability Matrix

This section shows two traceability matrices: between the modules and the requirements and between the modules and the anticipated changes. We should also consider documenting the mapping between these “abstract” modules and the Matlab files.

Req.	**Modules**
R1	M1, M2, M3, M9
R2	M2, M3
R3	M4
R4	M5, M9
R5	M5, M7, M9, M10, M11, M12
R6	M5, M7, M9, M10, M11, M12
R7	M5, M8, M9, M10, M12
R8	M5, M8, M9, M10, M12
R9	M6
R10	M5, M7, M9
R11	M5, M7, M8, M9

Table 2: Trace Between Requirements and Modules

AC	Modules
AC1	M1
AC2	M2
AC3	M3
AC4	M4
AC5	M5
AC6	M6
AC7	M7
AC8	M8
AC9	M9
AC10	M10
AC11	M11
AC12	M12

Table 3: Trace Between Anticipated Changes and Modules

7 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.

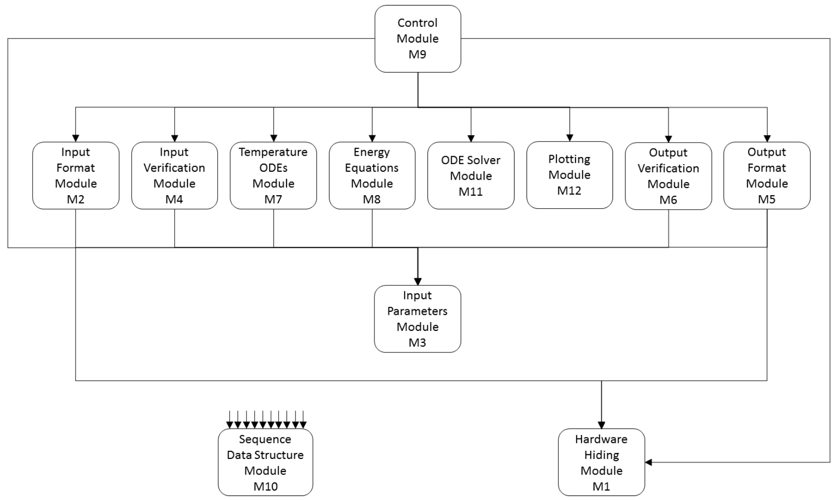

Figure 1: Use hierarchy among modules

References

D. L. Parnas, P. C. Clements, and D. M. Weiss. The modular structure of complex systems. In *ICSE '84: Proceedings of the 7th international conference on Software engineering*, pages 408–417, Piscataway, NJ, USA, 1984. IEEE Press. ISBN 0-8186-0528-6.

David L. Parnas. On the criteria to be used in decomposing systems into modules. *Comm. ACM, vol. 15, no. 2, pp. 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.

Greg Wilson, D.A. Aruliah, C. Titus Brown, Neil P. Chue Hong, Matt Davis, Richard T. Guy, Steven H.D. Haddock, Kathryn D. Huff, Ian M. Mitchell, Mark D. Plumblet, Ben Waugh, Ethan P. White, and Paul Wilson. Best practices for scientific computing. *CoRR*, abs/1210.0530, 2013.