Module Guide for SCEC (Solar cooker energy calculation)

Deesha Patel

March 16, 2023

1 Revision History

Date	Version	Notes
Mar 11, 2023	1.0	Initial release

2 Reference Material

This section records information for easy reference.

2.1 Abbreviations and Acronyms

symbol	**description**
AC	Anticipated Change
DAG	Directed Acyclic Graph
M	Module
MG	Module Guide
ODEs	Ordinary Differential Equations
OS	Operating System
R	Requirement
SC	Scientific Computing
SRS	Software Requirements Specification
SCEC	Solar Cooker Energy Calculator
UC	Unlikely Change

Contents

List of Tables

List of Figures

3 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 and Clements, February 1986). 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 implemented 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 4 lists the anticipated and unlikely changes of the software requirements. Section 5 summarizes the module decomposition that was constructed according to the likely changes. Section 6 specifies the connections between the software requirements and the modules. Section 7 gives a detailed description of the modules. Section 8 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 9 describes the use relation between modules.

4 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 4.1, and unlikely changes are listed in Section 4.2.

4.1 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 constraint on the input parameters.

AC4: How ODEs are calculated using input parameters.

AC5: How the energy equations are calculated using input parameters.

AC6: The defined time vector for ODEs.

AC7: The overall flow of the system.

AC8: The format of output data.

AC9: The algorithm to solve an ODE.

AC10: The implementation of plotting data.

4.2 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 source of input is always external to the software.

UC3: The outputs are displayed on the output device.

UC4: The goal of the system is to calculate the temperature of fluid.

UC5: The another goal of the system is to calculate the energy of the fluid.

UC6: The implemented ODEs are reading inputs from the file.

UC7: The implemented energy equation is reading inputs from the file.

5 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: Constant Value Module

M3: Energy Equation Module

M4: Input Format Module

M5: Input Parameter Module

M6: Output Format Module

M7: SCEC Control Module

M8: Temperature ODEs Module

M9: ODE Solver Module

M10: Plotting Result Module

M11: Sequence Data Structure Module

6 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.

Level 1	Level 2
Hardware-Hiding Module	
Behaviour-Hiding Module	Constant Value Module Energy Equation Module Input Format Module Input Parameter Module Output Format Module SCEC Control Module Temperature ODEs Module
Software Decision Module	ODE Solver Module Plotting Result Module Sequence Data Structure Module

Table 1: Module Hierarchy

7 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. *SCEC* means the module will be implemented by the SCEC 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.

7.1 Hardware Hiding Modules (M1)

Secrets: The data structure, algorithm and underlying hardware 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 SCEC system can use it to display outputs or to accept inputs.

Implemented By: OS

7.2 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: –

7.2.1 Constant Value Module (M2)

Secrets: The constant values used in the system.

Services: Storing all the constant values in this module and using it accordingly when needed to perform the operations.

Implemented By: SCEC

Type of Module: Record

7.2.2 Energy Equation Module (M3)

Secrets: The equation for solving the fluid energy using given inputs.

Services: Calculate the energy equation using the parameters in the input parameters module.

Implemented By: SCEC

Type of Module: Abstract Object

7.2.3 Input Format Module (M4)

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: SCEC

Type of Module: Abstract Object

7.2.4 Input Parameter Module (M5)

Secrets: The format and structure of the input parameters.

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

Implemented By: SCEC

Type of Module: Record

7.2.5 Output Format Module (M6)

Secrets: The format and structure of the output data.

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

Implemented By: SCEC

Type of Module: Abstract Object

7.2.6 SCEC Control Module (M7)

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

Services: Provides the main program to call other modules.

Implemented By: SCEC

Type of Module: Abstract Data Type

7.2.7 Temperature ODEs Module (M8)

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

Services: Defines the System of ODEs for solving the temperature values using initial values and input parameters module.

Implemented By: SCEC

Type of Module: Abstract Object

7.3 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: –

7.3.1 ODE Solver Module(M9)

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

Services: Provides solvers that can take different inputs like equation, initial conditions, time vector, numerical parameters, and solve it.

Implemented By: Python

Type of Module: Library

7.3.2 Plotting Result Module (M10)

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

Services: Provides plot function that can accept plotting data to plot the results using output module.

Implemented By: Python

Type of Module: Library

7.3.3 Sequence Data Structure Module (M11)

Secrets: The data structure for sequence data type.

Services: Provides different array operations including looping, adding and removing elements.

Implemented By: Python

Type of Module: Library

8 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, M7
R2	M4, M5, M7
R3	M6, M7, M8, M9, M10, M11
R4	M6, M7, M8, M9, M10, M11
R5	M3, M6, M7, M10, M11

Table 2: Trace Between Requirements and Modules

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

Table 3: Trace Between Anticipated Changes and Modules

9 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.

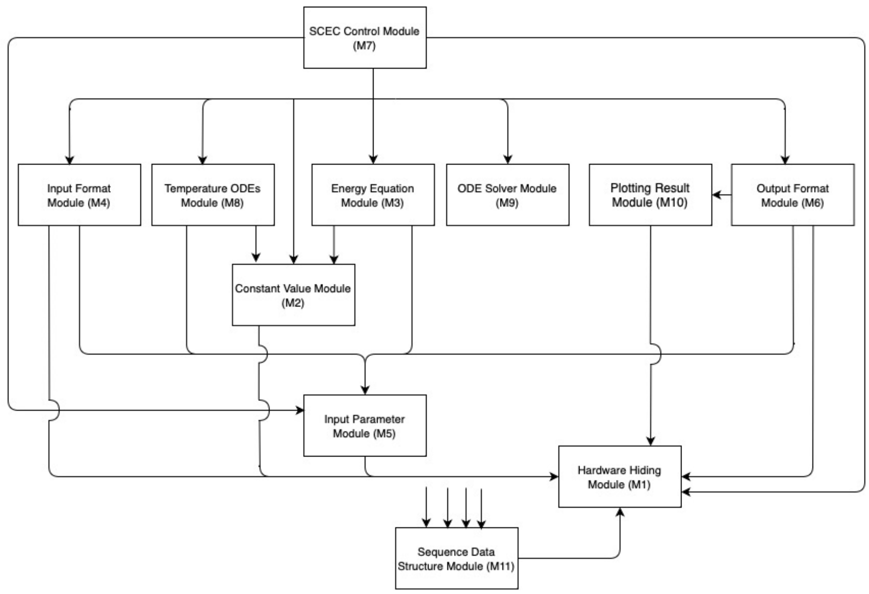

Figure 1: Use hierarchy among modules

References

D. L. Parnas, P. C. Clements, and D. M. Weiss. The modular structure of complex systems, 1984.

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.

David L. Parnas and P.C. Clements. A rational design process: How and why to fake it. *IEEE Transactions on Software Engineering*, 12(2):251–257, February 1986.