Module Guide for Minimization Cost Analysis for Data Centers

Ning Wang

March 18, 2023

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. In the best practices for scientific computing (SC), 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. 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, 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) . 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. Anticipated changes are numbered by **AC** followed by a number.

AC1: Increase or decrease in the number of data centers.

AC2: Changes in the total power consumption.

AC3: Changes in the distance between the power station and the data centers.

AC4: Changes in the maximum power consumption per data center.

AC5: Changes in the power loss due to distance.

AC6: The constraints on the output results.

AC7: Add more detailed comments to explain the purpose of each block of code and how it works.

AC8: Use more descriptive variable names to improve readability.

AC9: Add input validation to ensure that the user enters a valid number for the number of data centers.

AC10: Use a try-catch block to handle input errors and prevent the program from crashing.

AC11: The algorithm used for the minimization analysis.

AC12: Add a default value for the number of data centers in case the user does not enter a value.

AC13: Add error handling for cases where the number of data centers is too large or too small.

AC14: Add a function to calculate the power loss due to distance based on a formula rather than a constant value.

AC15: Add a function to calculate the cost for each data center based on the renewable energy rate, grid energy rate, and power consumption.

AC16: Add error handling for cases where the optimization fails to find a solution.

AC17: Add a function to check the total power consumption and display a warning if it is less than the assigned power for any data center.

AC18: Add a function to plot the power consumption and cost for each data center as a bar chart.

AC19: Add a function to save the power consumption and cost data to a file for further analysis.

AC20: Add a function to allow the user to specify the distance units (e.g., kilometers or miles).

AC21: Add a function to allow the user to choose between different optimization algorithms.

AC22: Add a function to allow the user to specify the tolerance for the total power consumption check.

AC23: Add a function to allow the user to compare different scenarios by varying the input parameters.

AC24: Add a function to generate a report summarizing the results and recommendations based on the optimization output.

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. Unlikely changes are numbered by **UC** followed by a number.

UC1: Input devices, only accept csv file, other type not looks like acceptable.

UC2: Remove the check for valid distance values between 0 and 2000.

UC3: Change the renewable energy rate to a fixed value instead of a variable input.

UC4: Remove the calculation of power loss due to distance.

UC5: Change the max load input to be less than the total load input.

UC6: Remove the lower bound constraint in the optimization.

UC7: Change the upper bound constraint for renewable energy consumption to be less than the max load.

UC8: Add a calculation for tax on the total cost.

UC9: The goal of the system is get the minimize cost.

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. Modules are numbered by **M** followed by a number.

M1: Hardware-Hiding Module

M2: Input File Reading Module

M3: Input Verification Module

M4: Output Format Module

M5: Output Verification Module

M6: Power loss Calculation Module

M7: Optimization Module

M8: Control Module

M9: Sequence Data Structure Module

M10: Plotting Module

Note that M1 is a commonly used module and is already implemented by the operating system. It will not be reimplemented. and M7 are already available in Matlab and will not be reimplemented.

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.

Level 1	**Level 2**
Hardware-Hiding Module	
Behaviour-Hiding Module	Input Parameters Module
	Input Verification Module
	Output Format Module
	Output Verification Module
	Control Module
	Specification Parameters Module
Software Decision Module	Optimization Module
	Sequence Data Structure Module
	Plotting Module

Table 1: Module Hierarchy

5 Module Decomposition

Modules are decomposed according to the principle of "information hiding" proposed by **?**. 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. *DCMCA* means the module will be implemented by the Data Centers Minimization Cost Analysis MATLAB Program. 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: To implement the virtual hardware use OS to display.

Services: –

Implemented By: OS

5.1.1 Input Parameter Module (M2)

Secrets: The CSV file containing input parameters, the system of files for input data.

Services: read$_{i}nput_{c}sv()functiontoreadtheinputCSVfileandreturnatableofinputparametersfromusers(incl$

5.1.2 Input Verification Module (M3)

Implemented By Secrets: The format and value under software constraints and under the capable requirement.

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: DCMCA

5.1.3 Output Format Module (M4)

Secrets: The format and structure of the output data.

Services: Outputs the results of the calculations, including the renewable and grid power allocated to each data center and total cost, then writes them into a CSV file.

Implemented By: DCMCA

5.1.4 Output Verification Module (M5)

Secrets: The algorithm used to approximate expected results.

Services: Verifies that the output results follow by checking if the sum of power assigned to each data center exceeds total power consumption.

Implemented By: DCMCA

5.1.5 Optimization Module (M7)

Secrets: The algorithms for solving the minimization cost analysis. The input parameters include the number of data centers, distances, rates of renewable and grid energy, and maximum and total power consumption.

Services: Defines the minimization equations set up and solves the linear programming problem to find the optimal power consumption.

Implemented By: MATLAB

5.1.6 Control Module (M8)

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

Services: Provides the main program.

Implemented By: DCMCA

5.1.7 Sequence Data Structure Module (M9)

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

5.1.8 Plotting Module (M10)

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

Services: Provides a plotted final graph to view the distribution of power consumption.

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. Three of the anticipated changes (AC2, AC3, AC4) related to the input parameters map to the same module (M2). The reason for this is that the services of this module will never need to be provided separately. Input will be provided to the system, stored and verified at the beginning of any simulation. From that point on, the only access needed to the input parameters is read access.

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

Table 2: Trace Between Requirements and Modules

AC	**Modules**
AC1	M1
AC2	M1
AC3	M1
AC4	M4
AC5	M4
AC6	M5
AC7	M6
AC8	M2
AC9	M2
AC10	M9
AC11	M8
AC12	M10
AC13	M8
AC14	M8
AC15	M8
AC16	M10
AC17	M2
AC18	M7
AC19	M7
AC20	M7
AC21	M9
AC22	M9
AC23	M10
AC24	M10

Table 3: Trace Between Anticipated Changes and Modules

7 Use Hierarchy Between Modules

In this section, the uses hierarchy between modules is provided. **?** 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.

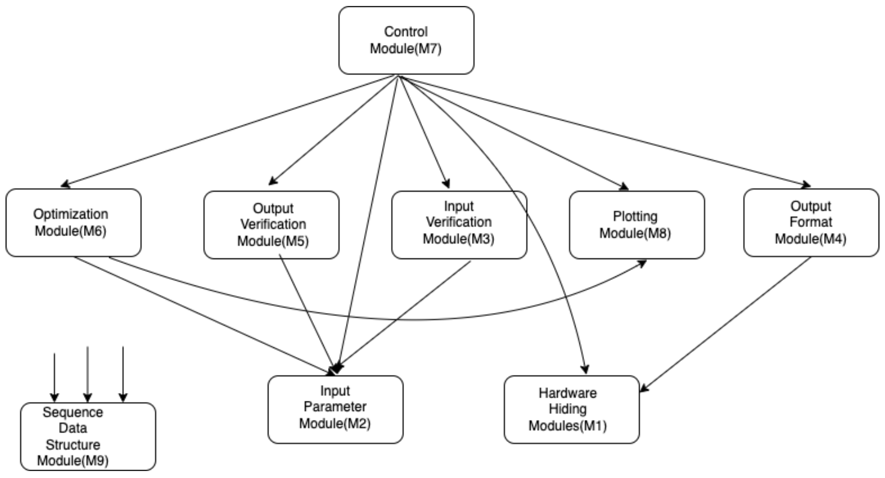
Control Module(M7)
Optimization Module(M6)
Output Verification Module(M5)
Input Verification Module(M3)
Plotting Module(M8)
Output Format Module(M4)
Sequence Data Structure Module(M9)
Input Parameter Module(M2)
Hardware Hiding Modules(M1)

Figure 1: Use hierarchy among modules