Module Guide for Measuring Microstructure Changes During Thermal Treatment

Team #30, ReSprint
Edwin Do
Joseph Braun
Timothy Chen
Abdul Nour Seddiki
Tyler Magarelli

January 18, 2023

1 Revision History

Date	Developer	Notes/Changes
Jan 15, 2023	Timothy Chen	Added Anticipated and Unlikey Cahnges (4)
Jan 15, 2023	Timothy	Added Module Hierarchy (5)
Jan 16, 2023	Edwin Do	Add use hierarchy diagram
Jan 18, 2023	Edwin Do	Add traceability matrices
Jan 18, 2023	Tyler Magarelli	Added Module Decomposition

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
OS	Operating System
R	Requirement
SC	Scientific Computing
SRS	Software Requirements Specification
Measuring Microstructure Changes During Thermal Treatment	Explanation of program name
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, 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 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 input data from the user.

AC3: The format of the output data.

AC4: The number of calculations the application will peform.

AC5: The number and type of graphs displayed.

AC6: The number of inputs the user can provide.

AC7: The criteria for validing data within the application.

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, Measurement Equipment).

AC8: Operating system (Winodws 10).

AC9: Calculation equation formulas (Resistence equation and Resistivity equation).

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: Input Communication Module

M3: Output Communication Module

M4: Remote Access Module

M5: Current State Module

M6: File Output Module

M7: Graphical Output Module

M8: Calculation Module

M9: User Input Validation Module

M10: Hardware Input Validation Module

Level 1	**Level 2**
Hardware-Hiding Module	
Behaviour-Hiding Module	Input Cummication Module Output Communication Module Remote Access Module Current State Module FileOutput Module Graphical Output Module
Software Decision Module	Calculation Module User Input Validation Module Hardware Input Validation Module

Table 1: Module Hierarchy

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.

7 Module Decomposition

7.1 Hardware Hiding Modules (M1)

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

Services: 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

7.2 Behaviour-Hiding Module

7.2.1 Communication Module

Secrets: Format input and output data and manage remote connection communication.

Services: Converts the data into the appropriate data structure used by the input/output parameters module. Controls communication for remote access.

Implemented By: -

Input Communication Module (M2)

Secrets: The format and structure of the input data (temperature, etc.)

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

Implemented By: Communication.cs

Type of Module: Abstract Data Type

Output Communication Module (M3)

Secrets: The format and structure of the output data.

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

Implemented By: Communication.cs

Type of Module: Abstract Data Type

Remote Access Module (M4)

Secrets: The implementation and algorithm of remote access. Login info and other credentials required for remote connection.

Services: Remote acces to the application for the user.

Implemented By: RemoteAccess.cs

Type of Module: Abstract Object Type

7.2.2 Display Module

Secrets: Display output data and and other information to the user.

Services: Receives data to be displayed and displays it on the screen for the user to see.

Implemented By: -

Current State Module (M5)

Secrets: The contents of obtaining current system state.

Services: Display state of hardware (current temp, current voltage, current sampling rate, etc.)

Implemented By: CurrentState.cs

Type of Module: Abstract Data Type

File Output Module (M6)

Secrets: The contents of obtaining correct files for output.

Services: Outputs the file the to screen for the user.

Implemented By: FileOutput.cs

Type of Module: Record

Graphical Output Module (M7)

Secrets: Obtaining the correct data to produce accurate graphs.

Services: Displays the data in a graphical format.

Implemented By: Graph.cs

Type of Module: Abstract Object Type

7.3 Software Decision Module

Calculation Module (M8)

Secrets: The mathematical equation for calculating resistivity and resistance.

Services: Solve the equation for resistivity and resistance respectively and return the result.

Implemented By: Calculation.cs

Type of Module: Abstract Data Type

7.3.1 Validation Module

Secrets: Validation criteria for various input streams.

Services: Validates the correctness of provided input from the given stream.

Implemented By: -

User Input Validation Module (M9)

Secrets: Validation criteria for user input.

Services: Validates the input from the user.

Implemented By: UserValidation.cs

Type of Module: Abstract Data Type

Hardware Input Validation Module(M10)

Secrets: Validation criteria for hardware input.

Services: Validates the input from the hardware.

Implemented By: HardwareValidation.cs

Type of Module: Abstract Data Type

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	M4, M5
R2	M6, M7, M8
R3	M2
R4	M6, M7, M8
R5	M4

Table 2: Trace Between Requirements and Modules

AC	**Modules**
AC1	M1
AC2	M2
AC3	M3
AC4	M8
AC5	M7
AC6	M2
AC7	M9

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.

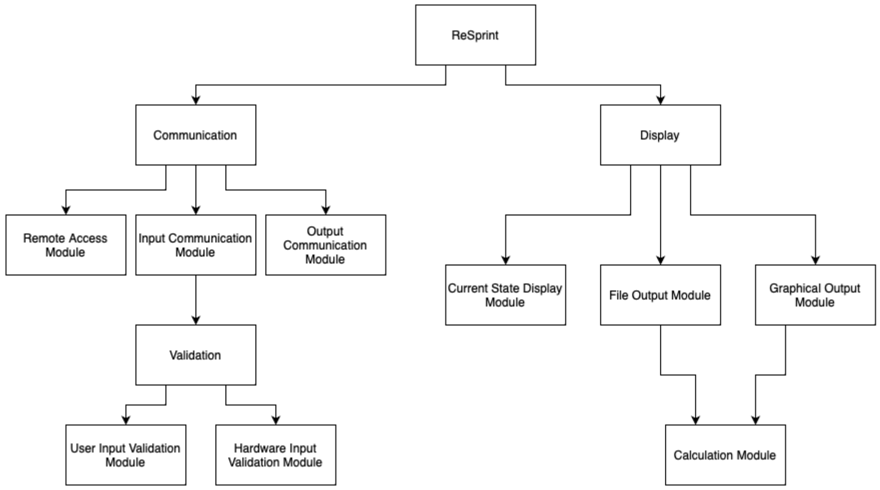

Figure 1: Use hierarchy among modules

References

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.