Module Guide for Measuring Microstructure Changes During Thermal Treatment

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

April 5, 2023

1 Revision History

Date	Developer	Notes/Changes
Jan 15, 2023	Timothy Chen	Added Anticipated and Unlikely Changes (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
Apr 2, 2023	Joseph Braun	Replaced .png graphic with .pdf version
Apr 5, 2023	Edwin Do	Updated revised module design

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 (**?**). 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 laid out by **?**, 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 (**?**). 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 the consistency among modules, feasibility of the decomposition, and flexibility of the design.

The rest of the document is organized as follows. Section **??** lists the anticipated and unlikely changes in the software requirements. Section **??** summarizes the module decomposition that was constructed according to the likely changes. Section **??** specifies the connections between the software requirements and the modules. Section **??** gives a detailed description of the modules. Section **??** 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 **??** 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 **??**, and unlikely changes are listed in Section **??**.

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 adopted 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 and hardware.

AC3: The number of calculations the application will peform.

AC4: The number and type of graphs displayed.

AC5: The number of inputs the user can provide.

AC6: The criteria for valiging 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 decisions 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).

UC2: Operating system (Windows 10).

UC3: Calculation equation formulas (Resistance 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 **??**. The modules listed below, which are leaves in the hierarchy tree, are the modules that will actually be implemented.

M1: Hardware-Hiding Module

M2: Current State Module

M3: File Output Module

M4: Graphical Output Module

M5: Calculation Module

M6: User Input Validation Module

M7: Instrument Input Validation Module

M8: Remote Access Module

Level 1	**Level 2**
Hardware-Hiding Module	
Behaviour-Hiding 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 **??**.

7 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. *Measuring Microstructure Changes During Thermal Treatment* means the module will be implemented by the Measuring Microstructure Changes During Thermal Treatment 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 (M??)

Note: This module is not implemented because changing the devices and the lab computer with the specialized ports is not likely to occur often. In addition, each device contains its own valid set of SCPI commands and will require a library to act as a wrapper. The effort of this may not be worthwhile since an update to the hardware will occur much less frequent and maintaining the library will require much more effort in comparison.

7.2 Behaviour-Hiding Module

7.2.1 Experiment Module

Responsible for all experiment related operations such as capturing data, displaying results and creating remote connections.

Secrets: The formatting of input and output data from both the user and instruments as well as management of remote access communication.

Services: Handles the data received from the hardware and formats it into the appropriate data structure to be used by other modules for validation. Controls communication for remote access.

Implemented By: -

Current State Module (M??)
Responsible for handling all state changes in the application from inputs.

Secrets: The contents of obtaining the current system state.

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

Implemented By: MainWindow.xaml.cs

Type of Module: Abstract Data Type

File Output Module (M??)
Responsible for constructing the output file using captured/calculated data for the user to access after the experiment has stopped.

Secrets: The contents of obtaining correct files for output.

Services: Outputs the file to a file location for the user in the format of CSV.

Implemented By: FileOutput.cs

Type of Module: Record

Graphical Output Module (M??)
Responsible for displaying a graphical representation of the data that is captured. Allow the user to modify the graphical representation with zoom/pan.

Secrets: The contents of 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

Remote Access Module (M??)
Responsible for creating a connection that can be accessed by remote clients. Remote clients can view experiment status and modify the current output.

Secrets: The implementation details of remote access (i.e. Remote URL, SSL Certificate).

Services: Remote access to the application for the user.

Implemented By: RemoteAccess.cs

Type of Module: Abstract Object Type

7.3 Software Decision Module

Calculation Module (M??)
Responsible for calculations of resistivity and resistance.

Secrets: The mathematical equation for calculating resistivity and resistance using unique constants for two types of temperature calculations.

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

Responsible for validating inputs from the user and hardware instruments.

Secrets: Validation criteria for various input streams.

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

Implemented By: -

User Input Validation Module (M??)
Responsible for validating inputs from the user.

Secrets: Validation criteria for user input.

Services: Validates the input from the user.

Implemented By: UserValidation.cs

Type of Module: Abstract Data Type

Instrument Input Validation Module(M??)
Responsible for validating inputs from the instruments (hardware).

Secrets: Validation criteria for hardware input.

Services: Validates the input from the hardware.

Implemented By: InstrumentInputValidation.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. The functional requirements can be found in the SRS document from the hyperlink above. FR6 (The installability of the application) does not correspond to a specific module.

Req.	Modules
FR1	M??, M??, M??
FR2	M??
FR3	M??
FR4	M??, M??
FR5	M??, M??
FR6	N/A

Table 2: Trace Between Requirements and Modules

AC	Modules
AC??	M??
AC??	M??, M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??

Table 3: Trace Between Anticipated Changes and Modules

9 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 **??** (Shown below) 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