Module Guide: FFT Library

Yuzhi Zhao

October 31, 2017

1 Revision History

Date	Version	Notes
Date 1	1.0	Notes
Date 2	1.1	Notes

Contents

List of Tables

List of Figures

2 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 layed out by **?**, 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 (**?**). 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 3 lists the anticipated and unlikely changes of the software requirements. Section 4 summarizes the module decomposition that was constructed according to the likely changes. Section 5 specifies the connections between the software requirements and the modules. Section 6 gives a detailed description of the modules. Section 7 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 8 describes the use relation between modules.

3 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 3.1, and unlikely changes are listed in Section 3.2.

3.1 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 format of the initial input data.

AC2: The format of the input parameters.

AC3: The constraints on the input parameters.

AC4: The formate of the final output data.

AC5: The algorithms for computing FFT.

AC6: The implementation for the array data structure.

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

UC3: The goal of the library is to do FFT calculation.

UC4: Whatever the input format is, they are all eventually transferred to datas to be computed.

4 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: FFT Calculation Module

M8: Control Module

M9: Sequence Data Structure Module

Level 1	**Level 2**
Hardware-Hiding Module	
Behaviour-Hiding Module	Input Parameters Module Input Verification Module Output Format Module Output Verification Module Temperature ODEs Module Energy Equations Module Control Module
Software Decision Module	FFT Calculation Module

Table 1: Module Hierarchy

5 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.

6 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. Also indicate if the module will be implemented specifically for the 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.

6.1 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

6.2 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 commonality analysis (CA) 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 CA.

Implemented By: –

6.2.1 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: FFTL

6.2.2 Input Parameters Module (M3)

Secrets: The format and structure of the input parameters.

Services: Stores the parameters needed for the program, including data types, radix information, and numerical parameters. The values can be read as needed. This module knows how many parameters it stores.

Implemented By: FFTL

6.2.3 Input Verification Module (M4)

Secrets: The format and structure of the software constraints.

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

Implemented By: FFTL

6.2.4 Output Format Module (M5)

Secrets: The format and structure of the output data.

Services: Outputs the results of the calculations, include the results of FFT based on the input datas.

Implemented By: FFTL

6.2.5 Output Verification Module (M6)

Secrets: The algorithm used to approximate expected results.

Services: Verifies that the output results follow the FFT calculation algorithm. Throws a warning if the output data number not equals the input data number.

Implemented By: FFTL

6.2.6 FFT Calculation Module (M7)

Secrets: The FFT calculation Algorithms to compute the output.

Services: Calculate the output based on input parameters module.

Implemented By: FFTL

6.2.7 Control Module (M8)

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

Services: Provides the main program.

Implemented By: FFTL

6.3 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 CA.

Services: Includes data structure and algorithms used in the system that do not provide direct interaction with the user.

Implemented By: –

6.3.1 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: C Programming Language

7 Traceability Matrix

This section shows two traceability matrices: between the modules and the requirements and between the modules and the anticipated changes.

Req.	Modules
IM1	M1, M2, M3, M4, M5, M6, M7, M8, M9
IM2	M1, M2, M3, M4, M5, M6, M7, M8, M9

Table 2: Trace Between Instance Models and Modules

AC	Modules
AC1	M2
AC2	M3
AC3	M4
AC4	M5
AC5	M7
AC6	M9

Table 3: Trace Between Anticipated Changes and Modules

8 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.

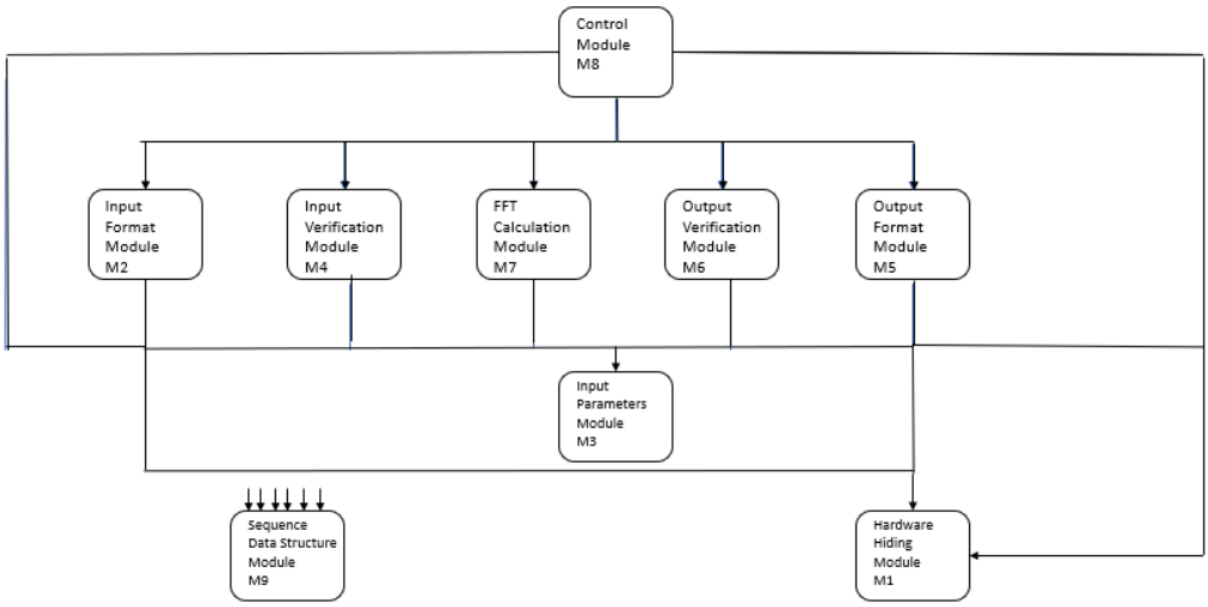

Figure 1: Use hierarchy among modules