Module Guide for FSL

Bo Cao

November 3, 2019

1 Revision History

Date	Version	Notes
Date 1	1.0	Notes
Date 2	1.1	Notes

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
FSL	Explanation of program name
UC	Unlikely Change
[etc. —SS]	[... —SS]

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 layed 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 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 and OS on which the software is running.

AC2: The format of the data that the conversion functions accept.

AC3: The type used in the API's languages that represents a function transformed into a CFS.

AC4: The linear solver used in the division operation.

AC5: The integral function used in transformation to CFS.

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: The data structure to store A_i's and B_i's in the CFS's.

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: The module that provides basic functions. Part of the implementing languages' run-time libraries.

M2: The linear solver module, implemented by others, and used by our library's division operation.

M3: The integral module, implemented by others, and used by the function in our library that transforms a mathematical function to a CFS.

M4: The data definition module.

M5: The conversion module to convert other data formats from/to CFS's.

M6: The transformation module to transform mathematical functions from/to CFS's

M7: The basic operation module including the addition, subtraction, multiplication, division and amplitude operation.

M8: The advanced operation module including the function of CFS, and tolerated equality operations.

Level 1	**Level 2**
Hardware-Hiding Module	M1 Basic functions module
Behaviour-Hiding Module	M4 Data definition module M5 Conversion module M6 Transformation module M7 Basic operations module M8 Advanced operations module
Software Decision Module	M2 Linear solver (external) M3 Integral (external)

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.

[The intention of this section is to document decisions that are made "between" the requirements and the design. To satisfy some requirements, design decisions need to be made. Rather than make these decisions implicit, they are explicitly recorded here. For instance, if a program has security requirements, a specific design decision may be made to satisfy those requirements with a password. In scientific examples, the choice of algorithm could potentially go here, if that is a decision that is exposed by the interface. —SS]

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. *FSL* means the module will be implemented by the FSL 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 Basic functions module (M1)

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

Services: Serves as the basic functions and data structure used by the rest of the system. This module provides the interface between the hardware, the OS and the software. So, the system can use it to build data structures, perform operations related to basic data types, display outputs or to accept inputs.

Implemented By: OS and run-time library of our implementing language.

7.2 Software Decision Modules

7.2.1 Linear solver module M2

Secrets: A linear equation solver.

Services: Solves the linear equation generated by FSL.

Implemented By: Third party.

7.2.2 Integral module M3

Secrets: A function that implements the integral of a function on a given interval.

Services: Calculates the integration in the transformation from a mathematical function to its CFS.

Implemented By: Third party

7.3 Behaviour-hiding modules

7.3.1 Data definition module M4

Secrets: Define the computational data structure of a CFS

Services: Provide CFS's getters and setters.

Implemented By: FSL

7.3.2 Conversion module M5

Secrets: Stores passed-in data in a CFS, and writes data in a CFS to user-designated spaces.

Services: Provide conversion from/to data in other formats.

Implemented By: FSL

7.3.3 Transformations M6

Secrets: Task1: Calculate the A_i's and B_i's in a CFS with given mathematical function $f(t)$, n, and ω, and store n, ω, A_i's and B_i's in a CFS.
Task2: Calculate function values based on variable values and function's CFS.

Services: Provide transformation from/to a mathematical function.

Implemented By: FSL

7.3.4 Basic operation M4

Secrets: Calculate the A_i's and B_i's for results of addition, subtraction, multiplication, and division operations, and calculate the result of amplitude operation.

Services: Provide addition, subtraction, multiplication, division and amplitude operations.

Implemented By: FSL

7.3.5 Advanced operation M4

Secrets: Calculate the A_i's and B_i's for functions of CFS's, and calculate the result of tolerated equality operation.

Services: Provide function of CFS and tolerated equality operations.

Implemented By: FSL

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	M??, M??, M??, M??
R2	M??, M??
R3	M??
R4	M??, M??
R5	M??, M??, M??, M??, M??, M??
R6	M??, M??, M??, M??, M??, M??
R7	M??, M??, M??, M??, M??
R8	M??, M??, M??, M??, M??
R9	M??
R10	M??, M??, M??
R11	M??, M??, M??, M??

Table 2: Trace Between Requirements and Modules

AC	**Modules**
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	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 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