Module Guide for SPDFM

S. Shayan Mousavi M.

November 16, 2020

1 Revision History

Date	Version	Notes
2020 11 10	1.0	Initial 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
SPDFM	Explanation of program name
UC	Unlikely Change
SPD	Surface Plasmon Dynamics

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 initial input data.

AC3: The type of the light source (pulsed sources instead of planewave).

AC4: The type of input geometry.

AC5: The method of inputting material properties.

AC6: Nonlocal hydrodynamic formulations.

AC7: Finite element method.

AC8: Internal data structure.

AC9: The format of the output data.

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).

...

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: SPDFM Control Module.

M3: Specification Parameters Module

M4: Input Parameters Module.

M5: Mesh Input Module.

M6: SPD Simulator Module.

M7: Frequency Domain Imaginary-PDE Solver Module.

M8: Frequency Domain Real-PDE Solver Module.

M9: Frequency Domain Solution Output Module.

M10: Data Structure Module.

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]

Level 1	Level 2
Hardware-Hiding Module	
Behaviour-Hiding Module	SPDFM Control Module
	Specification Parameters Module
	Input Parameters Modules
	Mesh Input Module
	Material Property Reading Module
	SPD Calculations Control Module
	Frequency Domain Solution Output Module
Software Decision Module	Mesh Generator Module
	Mesh Converter Module
	Frequency Domain Imaginary-PDE Solver Module
	Frequency Domain Real-PDE Solver Module
	Data Structure Module

Table 1: Module Hierarchy

7 Module Decomposition

Modules are decomposed according to the principle of "information hiding" proposed by Parnas et al. (1984). 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. *SPDFM* means the module will be implemented by the SPDFM 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 (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

7.2 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 specification (SRS) 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 SRS.

Implemented By: –

7.2.1 SPDFM Control Module(M2)

Secrets: Execution flow of SPDFM.

Services: Calls the different modules in the appropriate order.

Implemented By: SPDFM

7.2.2 Input Control Module(M4)

Secrets: Input flow of SPDFM.

Services: Calls the different input modules in the appropriate order.

Implemented By: SPDFM

Input Instruction Module. Geometry Reading Module. Light Source Reading Module. Material Property Reading Module.

7.2.3 Input Instruction Module(M5)

Secrets: Provide instruction flow of SPDFM.

Services: Providing a using a brief instruction of how to input data to the SPDFM.

Implemented By: SPDFM

7.2.4 Geometry Reading Module(M6)

Secrets: Inputs the meshed geometry.

Services: Loads, verifies and stores the input mesh data into the appropriate data structure.

Implemented By: SPDFM

7.2.5 Light Source Reading Module(M7)

Secrets: Input the light source related parameters.

Services: Loads, verifies and stores the input light source data into the appropriate data structure.

Implemented By: SPDFM

7.2.6 Material Property Reading Module(M??)

Secrets: Input the material properties.

Services: Loads, verifies and stores the input material properties data into the appropriate data structure.

Implemented By: SPDFM

7.2.7 Mesh Generator Module(M??)

Secrets: Generate the finite element mesh from the input geometry mesh.

Services: Uses the provided geometry .XML mesh to generate the finite element mesh space.

Implemented By: FEniCS and SPDFM

7.2.8 Mesh Coverts Module(M10)

Secrets: Convert the mesh data into the desired format for generating finite element mesh space.

Services: Converts the input .msh mesh into the .XML format mesh.

Implemented By: SPDFM

7.2.9 Finite Element Space Setup Module(M10)

Secrets: Define the appropriate finite element method function space .

Services: Converts the input .msh mesh into the .XML format mesh.

Implemented By: SPDFM

7.3 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 SRS.

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

Implemented By: –

7.3.1 Etc.

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	M1, 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
AC1	M1
AC2	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. 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.