Module Guide for SPDFM

S. Shayan Mousavi M.

December 2, 2020

1 Revision History

Date	Version	Notes
Nov 23 2020	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

expand your pragam name

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.

AC3: Format of the input mesh data.

AC4: Type of the finite element.

AC5: Finite element solver.

AC6: Internal data structure.

AC7: The format of the output data.

AC8: How the overall control of the calculations is orchestrated.

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).

AC8: Nonlocal surface plasmon hydrodynamic formulations.

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: Data Structure Module.

M4: Input Parameter Module.

M5: Constant Parameter Module.

M6: Mesh Input Module.

M7: SPD Simulator Module.

M8: Frequency Domain PDE Solver Module.

M9: Output Module.

Level 1	**Level 2**
Hardware-Hiding Module	
Behaviour-Hiding Module	SPDFM Control Module Input Parameter Modules Constan Parameters Module Mesh Input Module SPD Simulator Module Output Module
Software Decision Module	Frequency Domain PDE Solver Module Data Structure 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.

This is where you would mention the need to solve for the real & complex parts separately (alternatively, you could do this in the intro for the MIS)

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 Parameter Module (M4)

Secrets: Input required parameters for SPDFM to run the simulation.

Services: Inputting information required for setting up the light source (the boundary condition for solving PDE equations discussed in the SRS document), material properties of the different domains in the meshed geometry.

Implemented By: SPDFM

7.2.3 Constant Parameters Module (M5)

Secrets: The constant values used in the code.

Services: Storing all the constant values, including values mention in the table of specification parameters in the SRS document, for being called where needed at different modules.

Implemented By: SPDFM

7.2.4 Mesh Input Module (M6)

Secrets: Input mesh geometry.

Services: Inputting the meshed geometry. Converting the format of the input data (.mesh) to a readable format (.XML) for the PDE solver (FEniCS) used in SPDFM.

Implemented By: SPDFM

7.2.5 SPD Simulator Module (M7)

Secrets: Control flow of the different PDE solving units.

Services: Although at the current version of the software there is only one PDE solving module (M8), SPD Simulator module considers future expansions of the code. This module controls the flow of different PDE solving modules as some of them need to be run using results of other modules.

Implemented By: SPDFM

7.2.6 Mesh Output Module(M9)

Secrets: Output data.

Services: Storing the calculated electric field and current density in external vtk files.

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 Frequency Domain PDE Solver Module(M8)

Secrets: Solving the non-local hydrodynamic electromagnetic equations in the frequency domain

Services: Calculating electric field and current density at different frequencies.

Implemented By: SPDFM

7.3.2 Data Structure Module (M9)

Secrets: Data format for all parameters in SPDFM.

Services: Provides convenient format to store, read and manipulate all elements (pixel) from an image.

Implemented By: SPDFM

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	M3, M4, M6
R2	M3, M4, M6, M5
R3	M3, M7, M8
R4	M1, M3, M7, M8 M9

Table 2: Trace Between Requirements and Modules

AC	Modules
A1	M1
A2	M4
A3	M6
A4	M7, M8
A5	M8
A6	M3
A7	M9
A8	M2

Table 3: Trace Between Anticipated Changes and Modules

When you see one anticip change map to 2 or more modules you should explain this when introducing the table. Exceptions to the rule "one change maps to one module" are allowed, but they should be explained.

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.

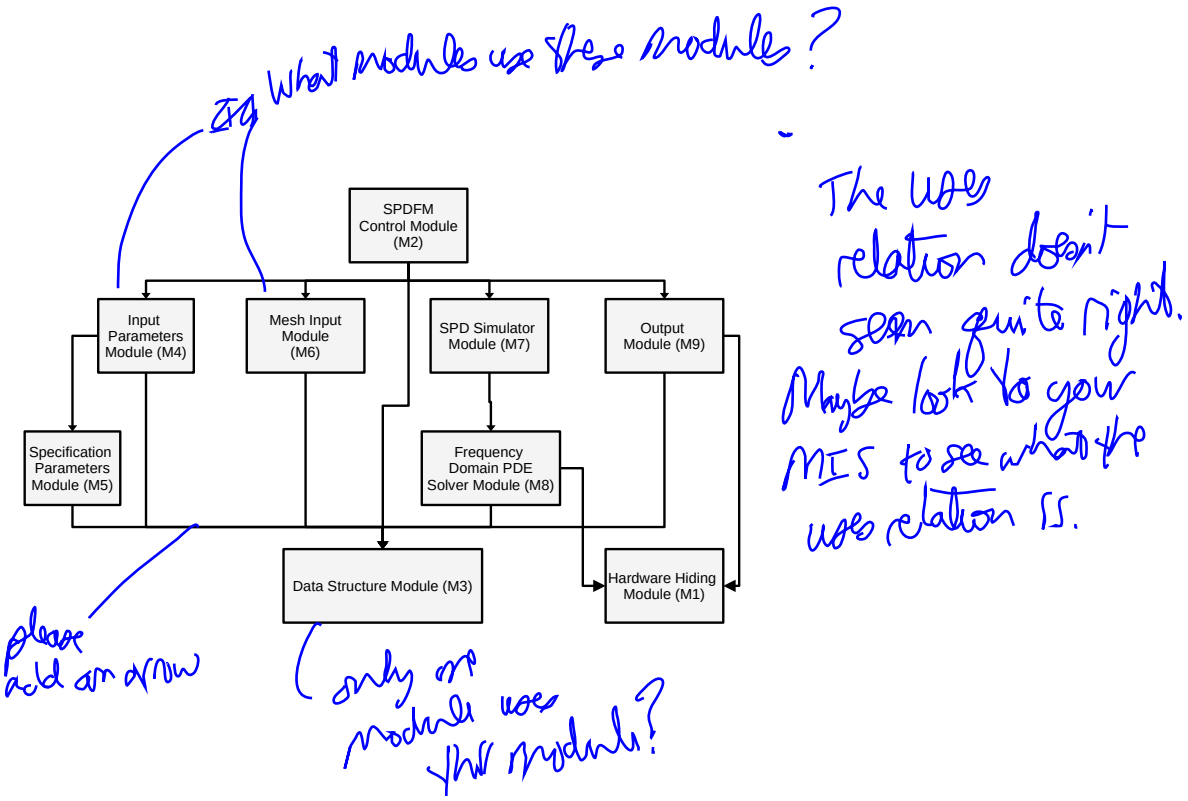

Figure 1: Use hierarchy among modules

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.