Module Guide: SpecSearch

Robert E. White

October 30, 2018

1 Revision History

Date	Version	Notes
October 26	1.0	Creation of first draft for presentation.

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 specific hardware on which the software is running.

AC2: The format of the initial input data.

AC3: The format of the output.

AC4: Format of the eigenfunction domain.

AC5: Numerical method of finding eigenvalues.

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: There will always be a two dimensional plot of the spectrum.

UC4: There will always be an error equation.

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 Parameters Module.

M3: Input Format Module.

M4: Output Format Module.

M5: Spectrum Matrix Module.

M6: Exact Lambda Equations Module.

M7: Lambda Error Equation Module.

M8: Domain Creation Module.

M9: Eigenvalue Solver Module.

M10: Eigenvector Solver Module.

M11: Diagonal Matrix Generator Module.

M12: Elliptic Integral Module.

M13: Elliptic Function Module.

M14: Plotting Module.

Level 1	Level 2
Hardware-Hiding Module	
Behaviour-Hiding Module	Input Parameters Module Input Format Module Output Format Module Spectrum Matrix Module Exact Lambda Equations Module Lambda Error Equation Module Domain Creation Module
Software Decision Module	Eigenvalue Solver Module Eigenvector Solver Module Diagonal Matrix Generator Module Elliptic Integral Module Ellipitic Function Module Plotting 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 Module (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 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: –

6.2.1 Input Parameters Module (M2)

Secrets: Names of the input parameters.

Services: Requests input from the user to be used by the Input format module.

Implemented By: SpecSearch

6.2.2 Input Format Module (M3)

Secrets: The format and structure of the input data.

Services: Converts the input data into a data structure for the other modules.

Implemented By: SpecSearch

6.2.3 Output Format Module (M4)

Secrets: The format and structure of the output data.

Services: Converts the output data into the data structure used by the plotting module.

Implemented By: SpecSearch

6.2.4 Operator Matrix Module (M5)

Secrets: The format and structure of the approximated operator matrix from the lax pair (see SRS). The entry values of the matrix operator and numerical method for approximating eigenfunction derivatives are also secrets.

Services: Creates the matrix meant to approximate the operator matrix from the lax pair (see SRS). This matrix will be used by the Eigenvalue and Eigenvector Solver Modules.

Implemented By: SpecSearch

6.2.5 Exact Lambda Equations Module (M6)

Secrets: The analytical expression for the two real eigenvalues.

Services: Calculates the two real eigenvalues from Segal et al. These two values will be used by the Lambda Error Equation Module.

Implemented By: SpecSearch

6.2.6 Lambda Error Equation Module (M7)

Secrets: The eigenvalue error equation.

Services: Calculates the difference between the the actual and theoretical eigenvalues from M7. This module will be used by the plotting module.

Implemented By: SpecSearch

6.2.7 Domain Creation Module (M8)

Secrets: The format and structure of the input data.

Services: Calculates the domain for which the eigenfunctions will be calculated. This module will be used by the operator matrix module.

Implemented By: SpecSearch

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 SRS.

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

Implemented By: –

6.3.1 Eigenvalue Solver Module (M9)

Secrets: The numerical algorithm for finding the eigenvalues of an n by n matrix.

Services: The eig MATLAB function finds the eigenvalues of an arbitrary n by n matrix.

Implemented By: SpecSearch

6.3.2 Eigenvector Solver Module (M10)

Secrets: The numerical algorithm for solving the eigenvalues of an n by n matrix.

Services: The eig MATLAB function finds the eigenvectors of an arbitrary n by n matrix.

Implemented By: SpecSearch

6.3.3 Diagonal Matrix Module (M11)

Secrets: The numerical algorithm for creating an n by n diagonal matrix.

Services: The diag MATLAB function creates an n by n diagonal matrix from a 1 by n vector. The diagonal entries correspond to the matrix. The diag function also creates a 1 by n vector from a diagonal matrix. This module will be used by the operator matrix module.

Implemented By: SpecSearch

6.3.4 Elliptic Integral Module (M12)

Secrets: The numerical algorithm for calculating the complete elliptic integral for some constant k.

Services: The elliptK MATLAB function calculates the integral of

$$\int_0^{\frac{\pi}{2}} \frac{dx}{\sqrt{1 - msin^2(x)}}$$

.

This module will be used by the Domain Creation Module.

Implemented By: SpecSearch

6.3.5 Elliptic Function Module (M13)

Secrets: The numerical algorithm for calculating the values of the jacobi elliptic functions.

Services: The ellipj MATLAB function calculates the values of dn,cn and sn for a particular domain. This module will be used by the operator matrix module.

Implemented By: SpecSearch

6.3.6 Plotting Module (M14)

Secrets: The plotting methods/algorithms.

Services: Creates a plot with prescribed input and output.

Implemented By: SpecSearch

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**
Rin	M1 , M2, M3
Rfind	M5, M7, M8, M9, M10, M11, M12,M13
Rcon	M8
Rplt	M14
Rstl	M14

Table 2: Trace Between Requirements and Modules

AC	**Modules**
AC1	M1
AC2	M3
AC3	M4
AC4	M5, M8
AC5	M9,M10

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

References