Module Guide: SpecSearch

Robert E. White

November 14, 2018

1 Revision History

Date	Version	Notes
October 26	1.0	Creation of first draft for presentation.
Nov 4	1.1	Post presentation edits. Feedback from Dr. Smith regarding hierarchy and control module.
Nov 7	1.2	Peer review edits.
Nov 9	1.3	Implementation of new MG template.

Contents

List of Tables

List of Figures

Abbreviations and Acronyms

symbol	**description**
MG	Module Guide
AC	Anticipated Change
SRS	System Requirements Specification
UC	Unlikely Change
M	Module
cn	Elliptic cosine function
sn	Elliptic sine function
dn	Elliptic delta function

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 (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 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 (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 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 (AC) 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.

AC6: Construction of Spectrum Matrix

AC7: Analytical calculation of eigenvalues.

AC8: Definition of error.

AC9: Numerical method algorithm.

AC10: Running speed (or data storage) standards.

AC11: Necessary accuracy of integral values.

AC12: Method used for approximating functions.

AC13: Method used for plotting data.

3.2 Unlikely Changes (UC)

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: The output (spectrum) is always displayed on the screen as a set of points on the complex plane. [Why is this an unlikely change? Why does the output have to go to the screen? It is an easy (and possibly useful change) to send the output to a file, or to make the output a table of values, rather than a plot. —SS]

UC3: The operator matrix and spectrum error can always be created from the inputs.

4 Module Hierarchy

This section provides an overview of the module design. Modules (M) 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.

M2: Input Parameters.

M3: Output Format.

M4: Spectrum Matrix.

M5: Exact Eigenvalue Equations.

M6: Spectrum Error Equation.

M7: Numerical Parameters.

M8: Eigenvalue and Eigenvector Solver.

M9: Diagonal Matrix Generator.

M10: Elliptic Integral.

M11: Elliptic Functions.

M12: Plotting.

M13: Linspace.

M14: Control.

Level 1	**Level 2**
Hardware-Hiding Module	
Behaviour-Hiding Module	Input Parameters Output Format Spectrum Matrix Exact Eigenvalue Equations Spectrum Error Equation Numerical Parameters Control
Software Decision Module	Eigenvalue and Eigenvector Solver Diagonal Matrix Generator Elliptic Integral Elliptic Functions Plotting Linspace

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 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. *SpecSearch* means the module will be implemented by the SpecSearch 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.

6.1 Hardware Hiding (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 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 (M2)

Secrets: The data structure for input parameters and how the values are verified.

Services: Gets input from the user, stores the input and verifies that the input variables satistfy [spell check —SS] the constraints in the SRS. Throws an error if any of the inputs violate a constraint and converts the input data into a data structure that is appropriate for the other modules.

Implemented By: SpecSearch

6.2.2 Output Format (M3)

Secrets: The format and structure of the output data.

Services: Converts the output data from the spectrum error module and eigenvalue solver module into a data structure necessary for the plotting module.

Implemented By: SpecSearch

6.2.3 Spectrum Matrix (M4)

Secrets: The structure of the spectrum matrix, its data entries, how it is created, and the numerical method for approximating its eigenfunctions.

Services: Creates the matrix that approximates the operator matrix from the lax pair (see SRS). [A more specific reference to the SRS would be great. —SS]

Implemented By: SpecSearch

6.2.4 Exact Eigenvalue Equations (M5)

Secrets: The analytical expression for the two real eigenvalues.

Services: Calculates the two purely real eigenvalues from literature for k. k is the elliptic parameter.

Implemented By: SpecSearch

6.2.5 Spectrum Error Equation (M6)

Secrets: The measure for error between exact and approximated eigenvalues.

Services: Calculates the absolute value of the difference between the the [proof read —SS] numerical (calculated with M9 and M5) and theoretical eigenvalues. [Do you need a specific module for this? Is there an algorithm here that justifies making this a secret? Why not relative error? Isn't this a one line formula in the code? Maybe I'll understand better when you write your MIS. —SS]

Implemented By: SpecSearch

6.2.6 Numerical Parameters (M7)

Secrets: The range of the eigenfunction domain, points in the periodic domain and equation for the numerical scaling factor that computes the eigenfunction derivatives. [How are these different from input? Instead of changing from execution to execution, are these values constant between runs? Are they like configuration parameters for the numerical algorithm? —SS]

Services: Creates the numerical parameters used for approximating the derivatives of the eigenfunctions.

Implemented By: SpecSearch

6.2.7 Control (M14)

Secrets: The algorithm that coordinates the overall program and interaction between modules.

Services: Is the main program.

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 and Vector Solver (M8)

Secrets: The numerical algorithm for calculating the eigenvalues and eigenvectors of an n by n matrix.

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

Implemented By: MATLAB

6.3.2 Diagonal Matrix (M9)

Secrets: The numerical algorithm for creating an n by n diagonal matrix from an n by 1 vector (and other way).

Services: The diag MATLAB function creates an n by n diagonal matrix from a 1 by n vector. The diagonal entries of the matrix are the elements of the vector. The diag function also creates a 1 by n vector from a diagonal matrix. [This is more of a function than a module. This is a service provided by the "matrix" module in MatLab. —SS]

Implemented By: MATLAB

6.3.3 Elliptic Integral (M10)

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

Services: The elliptK MATLAB function calculates the integral of

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

.

Implemented By: MATLAB

6.3.4 Elliptic Functions (M11)

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/vector.

Implemented By: MATLAB

6.3.5 Plotting (M12)

Secrets: The plotting methods/algorithms.

Services: Creates a two dimensional plot given a domain vector and a range vector of equal size.

Implemented By: MATLAB

6.3.6 Linspace (M13)

Secrets: The software algorithm for creating a vector with equally spaced entries.

Services: The linspace MATLAB function creates an array with prescribed endpoints and an equal difference between adjacent points. [I think this is more of a function than a module. —SS]

Implemented By: MATLAB

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 , M14
Rfind	M4, M7, M8, M9, M10, M11,M12, M14
Rcon	M12,M14,M5,M6
Rplt	M12,M14
Rstl	M13,M14

Table 2: Trace Between Requirements and Modules

AC	Modules
AC1	M1
AC2	M2
AC3	M3
AC4	M13
AC5	M8
AC6	M4
AC7	M5
AC8	M6
AC9	M7
AC10	M9
AC11	M10
AC12	M11
AC13	M12

Table 3: Trace Between Anticipated Changes and Modules

8 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.

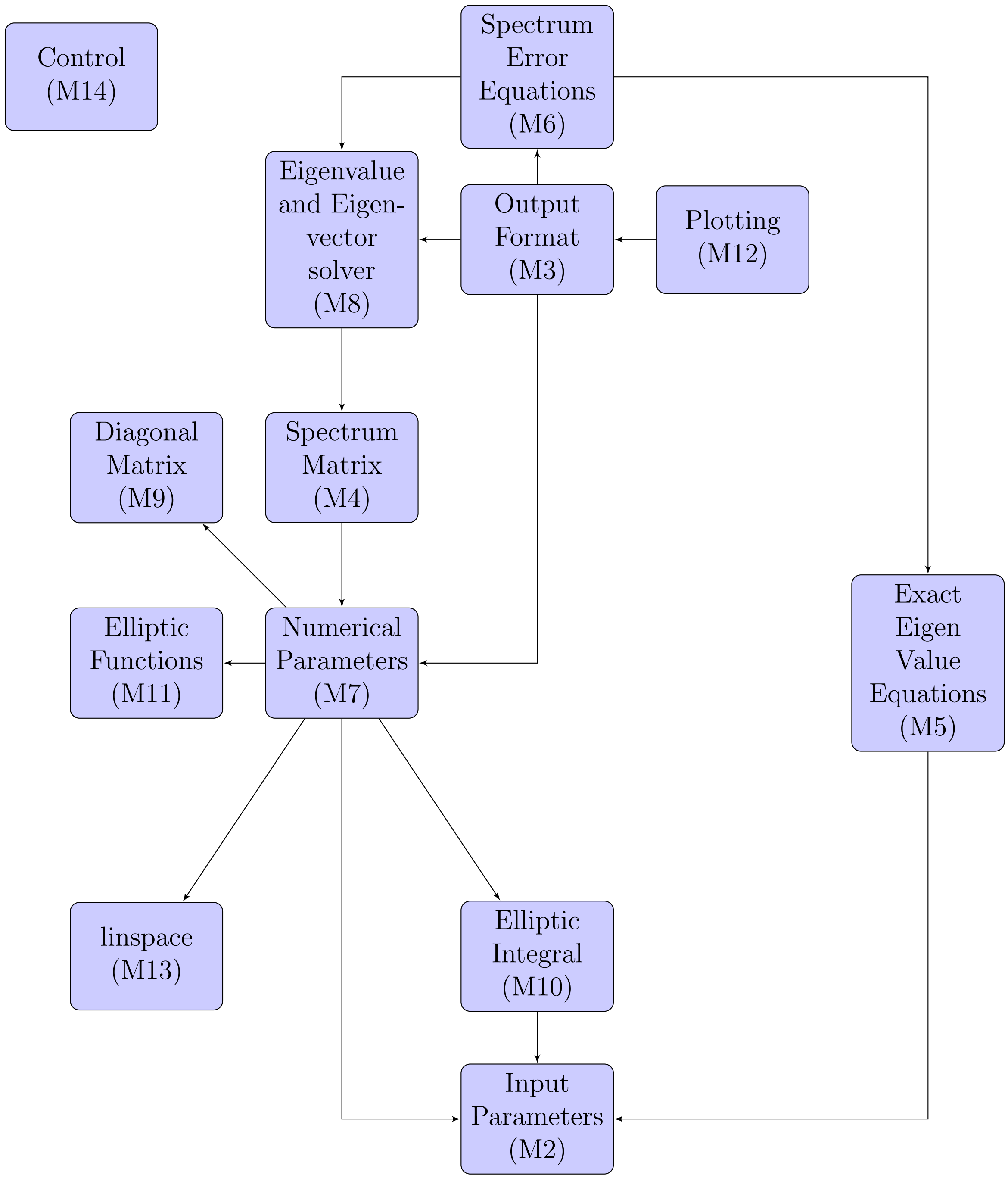

Figure 1: Use hierarchy among modules

[Nice use of tikz. —SS]

[Why doesn't the Control module use any other modules? Try to draw your uses relation so that all arrows are down. Why does numerical parameters use the Diagonal Matrix (and other services)? I had the impression that numerical parameters were configuration constants. Why does numerical parameters use input parameters? I'm sorry, but I really don't understand your design. —SS]

[Your bibliography didn't compile properly. I had to modify the path. I had to do the same thing for the Comments.tex file. —SS]

[Throughout this document you hard code in your references to other modules (M2, M9 etc.). You should make your life easier and use labels and references in LaTeX. —SS]

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.