Module Guide: SpectrumImageAnalysisPy

Isobel Bicket

December 17, 2017

1 Revision History

Date	Version	Notes
November 8, 2017	1.0	Initial draft

Contents

1 Revision History **i**

2 Introduction **1**

3 Anticipated and Unlikely Changes **2**
3.1 Anticipated Changes . . . 2
3.2 Unlikely Changes . . . 3

4 Module Hierarchy **4**

5 Connection Between Requirements and Design **5**

6 Module Decomposition **5**
6.1 Hardware Hiding Modules (M1) . . . 5
6.2 Behaviour-Hiding Module . . . 6
6.2.1 Import . . . 7
6.2.2 Import csv Module (M2) . . . 7
6.2.3 Import dm3 Module (M3) . . . 7
6.2.4 Import h5 Module (M4) . . . 7
6.2.5 Import rpl Module (M5) . . . 8
6.2.6 Export . . . 8
6.2.7 Export csv Module (M6) . . . 8
6.2.8 Export h5 Module (M7) . . . 8
6.2.9 Export png Module (M8) . . . 9
6.2.10 Export rpl Module (M9) . . . 9
6.2.11 Data Processing . . . 9
6.2.12 Data Processing: Richardson-Lucy Deconvolution (M10) . . . 9
6.2.13 Data Processing: Normalization (M11) . . . 9
6.2.14 Data Processing: Gain Correction (M12) . . . 10
6.2.15 Data Processing: Background Correction (M13) . . . 10
6.2.16 Data Extraction . . . 10
6.2.17 Data Extraction: 1D Slice (M14) . . . 10
6.2.18 Data Extraction: 2D Mask (M15) . . . 11
6.2.19 Data Extraction: 3D Mask (M16) . . . 11
6.2.20 Display . . . 11
6.2.21 Display: 1D Spectrum Plot (M17) . . . 11
6.2.22 Display: 2D Image Plot (M18) . . . 11
6.2.23 Display: 3D Spectrum Image Plot (M19) . . . 12
6.3 Software Decision Module . . . 12
6.3.1 Data . . . 12
6.3.2 Data: 1D Spectrum (M20) . . . 12
6.3.3 Data: 2D Image (M21) . . . 12

6.3.4 Data: 3D Spectrum Image (M22) 13
6.3.5 Array Data Structure 13
6.3.6 Plotting Library 13

7 Traceability Matrix **13**

8 Use Hierarchy Between Modules **15**

List of Tables

1 Module Hierarchy 6
2 Trace Between Requirements and Modules 13
3 Trace Between Anticipated Changes and Modules 14

List of Figures

1 Use Hierarchy for Level 1 and Level 2 modules 15
2 Use Hierarchy for Level 3 Import modules 16
3 Use Hierarchy for Level 3 Export modules 16
4 Use Hierarchy for Level 3 Data Processing modules 16
5 Use Hierarchy for Level 3 Data Extraction modules 17
6 Use Hierarchy for Level 3 Display modules 17
7 Use Hierarchy for Level 3 Data modules 18

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 [1]. We advocate a decomposition based on the principle of information hiding [2]. 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 [1], 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 [1]. 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 headings or data format of saved spectrum csv files.

AC3: The data type inside dm3 files.

AC4: The locations/labels of data and metadata inside dm3 files. [Having two secrets for dm3 files (and some of the others) seems like too fine a subdivision to me. I'm willing to be convinced that it is necessary, but the "format of the file" is usually adequate as an anticipated change (and secret). —SS]

AC5: Expanding support to dm4 files.

AC6: The data type inside h5 files.

AC7: The locations/labels of data and metadata inside h5 files.

AC8: Adding new data processing algorithms.

AC9: Functions used for RL deconvolution (optimizations for reduced error or speed).

AC10: Optimizations on normalization algorithm. [I think the anticipated change is the normalization algorithm. The purpose of the change may be optimization of the algorithm, but the change is the algorithm. A similar comment applies for some of the other anticipated changes. —SS]

AC11: Optimizations on gain correction algorithm.

AC12: Optimizations on background correction algorithm.

AC13: Format of 1D spectrum plot.

AC14: Format of 2D image plot.

AC15: Layout of 3D spectrum image plotting display.

AC16: Optimizations for faster response of spectrum image plotting display.

AC17: Changes to the interactive commands in the spectrum image plotting display.

AC18: Units or label on the spectral axis.

AC19: Method used to slice spectra.

AC20: Method of masking 2D images.

AC21: Method of masking 3D images.

AC22: Adding new modules for image stacks (4D datasets).

AC23: New file format for exported image files (*e.g.*, change to tiff).

AC24: New file format for exporting spectrum image files.

AC25: Format changes for exporting spectrum image files to h5.

AC26: Format changes for exporting spectrum image files to rpl.

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 decisions 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 hardware devices (a keyboard and mouse for input, a monitor screen for output).

UC2: The nature of the hardware used to run the software (a laptop or desktop computer containing sufficient memory and computing power to perform the operations required).

UC3: There will always be a source of input data external to the software.

UC4: The use of arrays for holding data. [This is an unusual (and I believe unnecessary) restriction. Mathematically (abstractly) what you need is a sequence. If it is a numpy array, or a Python list, or another data structure in another language, you could still implement your software. —SS]

UC5: Plotting library used.

UC6: Array structure library used. [This seems like an unnecessary restriction to me. —SS]

UC7: Format of 1D data (y-axis and x-axis data). [This also seems like an unnecessary restriction. —SS]

UC8: Format of 2D data ((x, y) data array).

UC9: Format of 3D data ((x, y, E) data array).

UC10: The order of the spatial and spectral axes in a spectrum image.

UC11: The data type to export.

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.

M2: Import .csv.

M3: Import .dm3.

M4: Import .h5.

M5: Import .rpl.

M6: Export .csv.

M7: Export .h5.

M8: Export .png.

M9: Export .rpl.

M10: Data Processing Richardson-Lucy Deconvolution.

M11: Data Processing Normalization.

M12: Data Processing Gain Correction.

M13: Data Processing Background Correction.

M14: Data Extraction 1D Slice.

M15: Data Extraction 2D Mask.

M16: Data Extraction 3D Mask.

M17: Display 1D Spectrum.

M18: Display 2D Image.

M19: Display 3D Spectrum Image.

M20: Data 1D Spectrum.

M21: Data 2D Image.

M22: Data 3D Spectrum Image.

M23: Array Data Structure. [The more abstract notion of sequence hiding might be better here. The choice of using arrays in numpy is for efficiency, but a sequence ADT with an interface that provides the services you need should be fine, and it would be more amenable to change. —SS]

M24: Plotting Library.

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 [1]. 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 Modules (M1)

Secrets: The data structure and algorithm used to implement the virtual hardware.

Level 1	Level 2	Level 3
Hardware-Hiding Module		
Behaviour-Hiding Module	Import	csv dm3 h5 rpl
	Export	csv h5 png rpl
	Data processing	Richardson-Lucy Deconvolution Normalization Gain correction Background correction
	Data extraction	1D slice 2D mask 3D mask
	Display	1D spectrum plot 2D image plot 3D spectrum image plot
Software Decision Module	Data	Spectrum Image Spectrum Image
	Array Data Structure	
	Plotting Library	

Table 1: Module Hierarchy

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 Import

Secrets: Container for modules concerning reading data from files (unifies the interface to the Level 3 import modules).

Services: Imports spectrum, image, or spectrum image data from various formats, as defined in the Level 3 modules to follow (M2, M3, M4, M5).

Implemented By: –

6.2.2 Import csv Module (M2)

Secrets: Import from csv (*e.g.*, as created by SpectrumImageAnalysisPy).

Services: Reads data from a csv file (which must be formatted appropriately) and assigns the data to the spectrum module. Throws an error if the format of the csv file is wrong.

Implemented By: SpectrumImageAnalysisPy

6.2.3 Import dm3 Module (M3)

Secrets: Import data from Gatan Digital Micrograph file (dm3).

Services: Reads data from a dm3 file and assigns the data to the spectrum, spectrum image, or image data module, as appropriate based on the dimensions of the data. Will assign calibrations based on the metadata in the file. Throws an error if the data type is not one of the expected types or the format is not one of the expected formats.

Implemented By: SpectrumImageAnalysisPy

6.2.4 Import h5 Module (M4)

Secrets: Import data from an h5 file (*e.g.*, as produced by Odemis CL acquisition software, or by SpectrumImageAnalysisPy).

Services: Reads data from an h5 file and assigns the data to the spectrum, spectrum image, or image data module, as appropriate based on the dimensions of the data. Will assign calibrations based on the metadata in the file. Throws an error if the data type is not one of the expected types or the format is not one of the expected formats.

Implemented By: SpectrumImageAnalysisPy

6.2.5 Import rpl Module (M5)

Secrets: Import data from rpl file (*e.g.*, as exported by SpectrumImageAnalysisPy)

Services: Reads data from a rpl file and assigns the data to the spectrum, spectrum image, or image data module, as appropriate based on the dimensions of the data. Will assign calibrations based on the metadata in the file. Throws an error if the data type is not one of the expected types or the format is not one of the expected formats.

Implemented By: SpectrumImageAnalysisPy

6.2.6 Export

Secrets: Container for modules concerning exporting data to files.

Services: Writing data (spectrum, image, or spectrum image) to various formats, as defined by Level 3 modules (M6, M7, M8, M9).

Implemented By: –

6.2.7 Export csv Module (M6)

Secrets: Export data to csv file.

Services: Takes spectrum data, formats it as appropriate, and exports the spectrum range (x-axis) and the intensity (y-axis) as a comma separated value file to the filepath specified by the user. Checks that the data is the appropriate dimensions (a 1D dataset, with x-axis values and corresponding y-axis values).

Implemented By: SpectrumImageAnalysisPy

6.2.8 Export h5 Module (M7)

Secrets: Export data to h5 file.

Services: Takes image, spectrum, or spectrum image data, formats it for writing to a file, and exports the spectrum range, calibrations, metadata, and the spectrum image data to an h5 file. Verifies that the data is the correct dimensionality and format before exporting.

Implemented By: SpectrumImageAnalysisPy

6.2.9 Export png Module (M8)

Secrets: Export image data to png file.

Services: Takes image data, formats it as appropriate, and writes it to a png file, with scalebar if requested by the user. Applies the colourmap or transparency as requested by the user. Verifies that the data is 2D, and will convert intensity values to those appropriate for an image file (0-255).

Implemented By: SpectrumImageAnalysisPy

6.2.10 Export rpl Module (M9)

Secrets: Export spectrum image data to rpl file.

Services: Takes spectrum image data, formats it as appropriate, and writes it to a rpl file, including any metadata or calibrations which are known. Verifies the data is the correct dimensionality and format before exporting.

Implemented By: SpectrumImageAnalysisPy

6.2.11 Data Processing

Secrets: Container for data processing modules.

Services: Performs data processing functions, as defined by Level 3 modules (M10, M11, M12).

Implemented By: –

6.2.12 Data Processing: Richardson-Lucy Deconvolution (M10)

Secrets: Richardson-Lucy deconvolution algorithm.

Services: Performs Richardson-Lucy deconvolution on the desired spectrum or spectrum image input, and returns a deconvolved spectrum or spectrum image. Requires the input of a point spread function, a spectrum or spectrum image, and the number of iterations to perform.

Implemented By: SpectrumImageAnalysisPy

6.2.13 Data Processing: Normalization (M11)

Secrets: Normalization algorithm.

Services: Performs normalization of the spectrum or spectrum image, as desired by the user. User can input the channel or a range of channels on which to normalize. Returns a normalized spectrum or spectrum image. Checks for divide-by-zero errors and will not perform the normalization if this is an issue.

Implemented By: SpectrumImageAnalysisPy

6.2.14 Data Processing: Gain Correction (M12)

Secrets: Gain correction algorithm.

Services: Performs gain correction on the spectrum or spectrum image input. A gain reference is also required. Returns a spectrum or spectrum image which has been corrected for gain.

Implemented By: SpectrumImageAnalysisPy

6.2.15 Data Processing: Background Correction (M13)

Secrets: Background correction algorithm.

Services: Performs background correction on input spectrum or spectrum image and returns the corrected spectrum or spectrum image. A background reference is required. Returns a spectrum or spectrum image which has been corrected for the background.

Implemented By: SpectrumImageAnalysisPy

6.2.16 Data Extraction

Secrets: Contains modules for extracting segments of data.

Services: Extract 1D, 2D, and 3D segments of data, as desired by the user, defined in Level 3 modules (M14, M15, M16).

Implemented By: –

6.2.17 Data Extraction: 1D Slice (M14)

Secrets: Perform 1D slice on the spectrum axis, based on input from the user.

Services: Takes input on the desired spectral range to slice, performs the slice on the data, and returns the sliced range.

Implemented By: SpectrumImageAnalysisPy

6.2.18 Data Extraction: 2D Mask (M15)

Secrets: Perform 2D mask operation on the image axis, based on input from the user.

Services: Takes input on the desired spatial pixels to mask and returns the area under the mask.

Implemented By: SpectrumImageAnalysisPy

6.2.19 Data Extraction: 3D Mask (M16)

Secrets: Perform 3D mask operation on a spectrum image, based on input from the user.

Services: Takes input from a 2D mask and extends this mask along the spectral axis, or takes input from a 1D slice and extends this slice along the spatial axes to produce a 3D mask. Returns the volume under the mask.

Implemented By: SpectrumImageAnalysisPy

6.2.20 Display

Secrets: Display a spectrum, image, or spectrum image for the user.

Services: Plots a 1D spectrum, plots a 2D image, or plots a 3D spectrum image, depending on the input data.

Implemented By: –

6.2.21 Display: 1D Spectrum Plot (M17)

Secrets: Displays a 1D spectrum plot for the user.

Services: Takes in a 1D spectrum and plots it for the user with the appropriate x- and y- axis labels and units. Allows the user to plot multiple spectra on one plot with a defined colourmap.

Implemented By: SpectrumImageAnalysisPy

6.2.22 Display: 2D Image Plot (M18)

Secrets: Displays 2D image plot for the user.

Services: Takes in 2D image data and plots it for the user to interact with, including a scalebar if desired by the user. Allows user to define a colourmap.

Implemented By: SpectrumImageAnalysisPy

6.2.23 Display: 3D Spectrum Image Plot (M19)

Secrets: Displays 3D spectrum image for the user.

Services: Combines a 1D spectrum plot and 2D image plot to allow the user to visualize and navigate a 3D data set.

Implemented By: SpectrumImageAnalysisPy

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 Data

Secrets: Contains the 1D, 2D, and 3D datasets.

Services: Includes the structure of the datasets for spectra (1D), images (2D), and spectrum images (3D), defined in Level 3 modules (M20, M21, M22). Includes spatial and spectral calibrations and metadata as appropriate.

Implemented By: –

6.3.2 Data: 1D Spectrum (M20)

Secrets: Holds data structure for 1D spectrum data.

Services: Contains the data for a spectrum, including x-axis values and y-axis values, and any relevant calibrations or metadata.

Implemented By: SpectrumImageAnalysisPy

6.3.3 Data: 2D Image (M21)

Secrets: Holds data structure for 2D image data.

Services: Contains the data for an image, including (x, y) values, and any relevant spatial calibrations or metadata.

Implemented By: SpectrumImageAnalysisPy

6.3.4 Data: 3D Spectrum Image (M22)

Secrets: Holds data structure for 3D spectrum image data.

Services: Contains the data for a spectrum image, including the values along the spectral axis, and intensity values for the whole volume defined, as well as any relevant calibrations or metadata.

Implemented By: SpectrumImageAnalysisPy

6.3.5 Array Data Structure

Secrets: Data structure for arrays.

Services: Functions and support for array data types.

Implemented By: Python

6.3.6 Plotting Library

Secrets: Library for creating plots.

Services: Building blocks for creating 1D, 2D, and 3D plots.

Implemented By: Python

7 Traceability Matrix

This section shows two traceability matrices: between the modules and the requirements and between the modules and the anticipated changes.

Requirement	Modules
SI input	M1, M3, M4, M22
Spectrum input	M1, M2, M20
Input dimensions	M2, M3, M4, M5
SI slicing	M8, M14, M16, M17, M18, M19
SI area	M6, M15, M16, M17, M18, M19
Deconvolution	M10, M19, M20, M22
Normalization	M11, M14, M22
Background	M13, M20, M22
Gain	M12, M20, M22

Table 2: Trace Between Requirements and Modules

Ideally, each anticipated change should correspond to one module. There is one anticipated change to SpectrumImageAnalysisPywhich may affect multiple modules: AC2. The modules affected by this change are interdependent on each other, concerning the reading and writing of csv files. The format which must be read corresponds to output from a previous use of SpectrumImageAnalysisPy, so must correspond to the format which was written beforehand. Combining the import and export modules, however, will result in cyclic paths in the Use Hierarchy, as Export uses Data Extraction, which uses Data, which uses Import, so the decision was made to keep the Import and Export modules separate from each other.

Where no leaf module was relevant to the anticipated change, the name of a node module is written. This might happen when the anticipated change results in the addition of a new module under that node.

AC	**Modules**
AC1	M1
AC2	M2, M6
AC3	M3
AC4	M3
AC5	Import
AC6	M4
AC7	M4
AC8	Data Processing
AC9	M10
AC10	M11
AC11	M12
AC12	M13
AC13	M17
AC14	M18
AC15	M19
AC16	M19
AC17	M19
AC18	M20
AC19	M14
AC20	M15
AC21	M16
AC22	Data
AC23	Export
AC24	Export
AC25	M7
AC26	M9

Table 3: Trace Between Anticipated Changes and Modules

8 Use Hierarchy Between Modules

In this section, the uses hierarchy between modules is provided. [3] 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 Level 1 and 2 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. For clarity, the Level 3 modules have been split apart from the main Use Hierarchy diagram and are shown in more detail in Figure 2, Figure 3, Figure 4, Figure 5, Figure 6, and Figure 7.

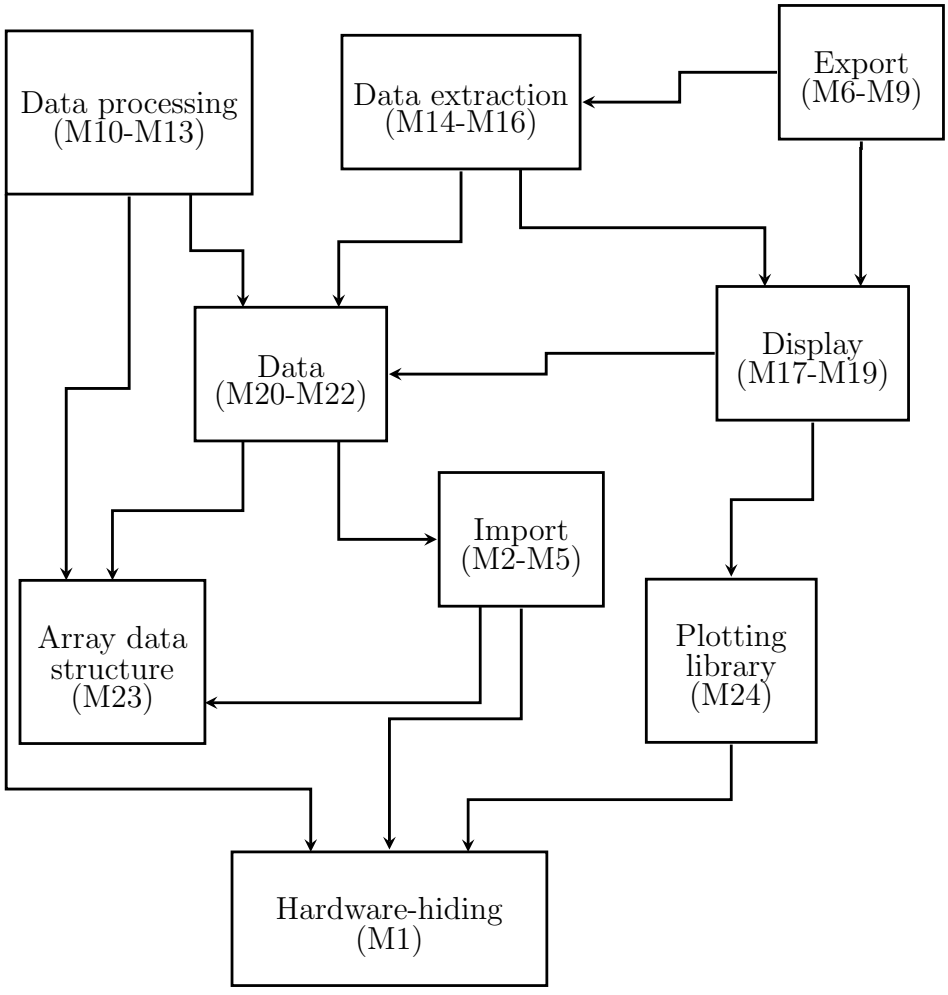

Figure 1: Use Hierarchy for Level 1 and Level 2 modules

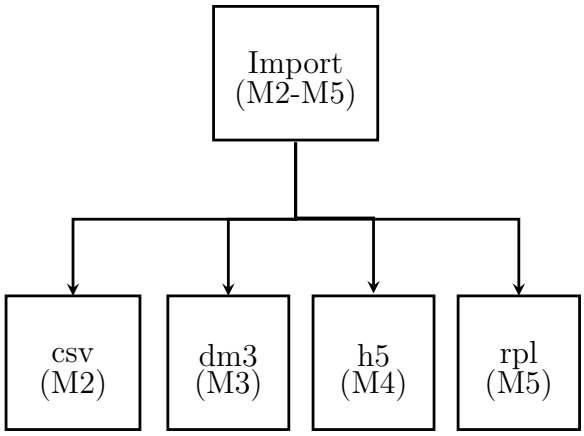

Figure 2: Use Hierarchy for Level 3 Import modules

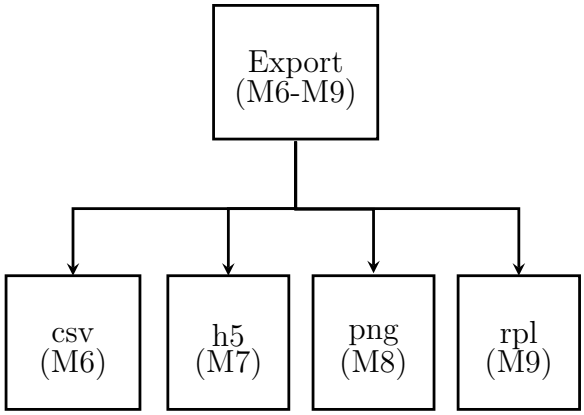

Figure 3: Use Hierarchy for Level 3 Export modules

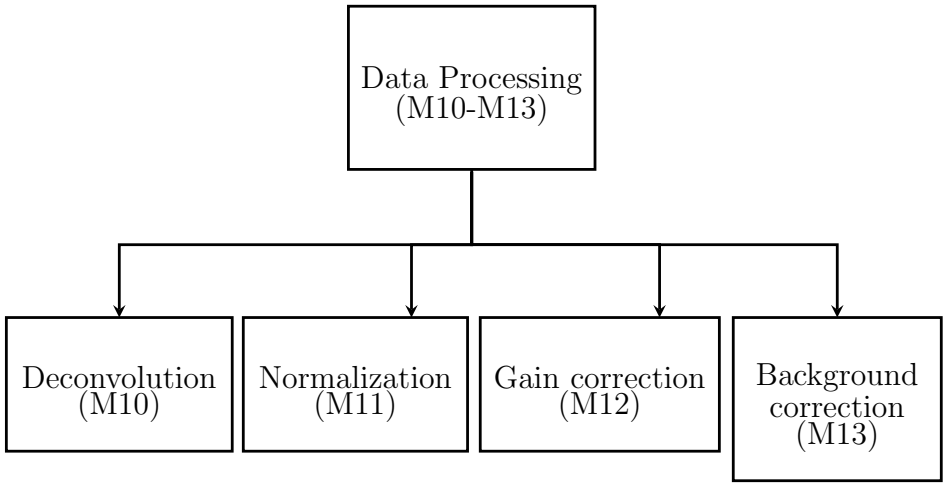

Figure 4: Use Hierarchy for Level 3 Data Processing modules

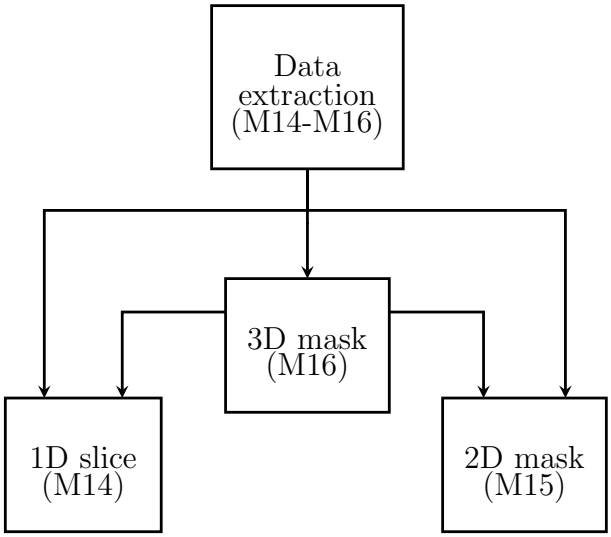

Figure 5: Use Hierarchy for Level 3 Data Extraction modules

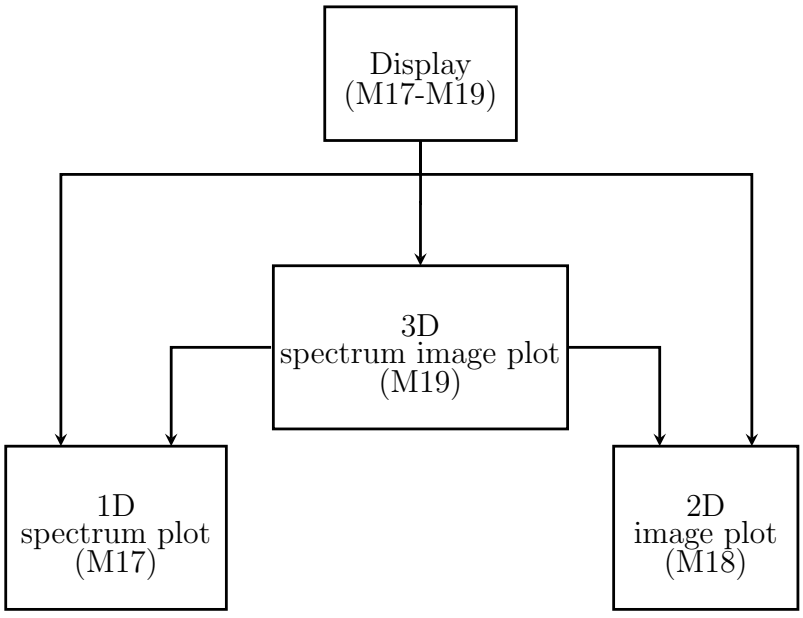

Figure 6: Use Hierarchy for Level 3 Display modules

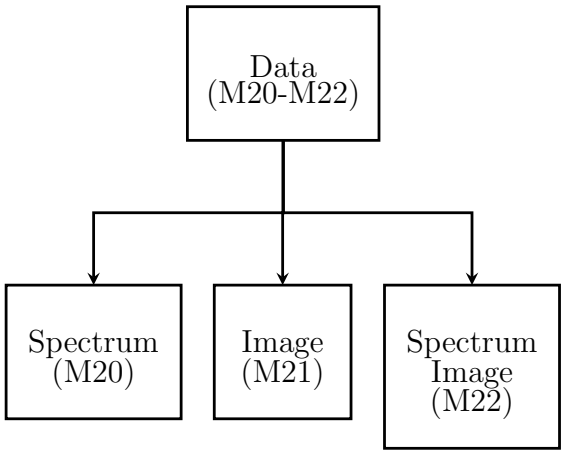

Figure 7: Use Hierarchy for Level 3 Data modules

[Your design looks like a great start. It will be easier to tell how well the pieces fit together as you work on your MIS. If anything becomes awkward, or too detailed, remember that you can use abstraction to simplify the interfaces, and maybe reduce your number of modules. —SS]

References

[1] D. L. Parnas, P. C. Clements, and D. M. Weiss, "The Modular Structure of Complex Systems," *IEEE Transactions on Software Engineering*, vol. SE-11, pp. 259–266, Mar. 1985.

[2] D. L. Parnas, "On the criteria to be used in decomposing systems into modules," *Comm. ACM*, vol. 15, pp. 1053–1058, December 1972.

[3] D. L. Parnas, "Designing software for ease of extension and contraction," in *ICSE '78: Proceedings of the 3rd international conference on Software engineering*, (Piscataway, NJ, USA), pp. 264–277, IEEE Press, 1978.