Module Guide: Project Title

Author Name

November 8, 2017

1 Revision History

Date	Version	Notes
Date 1	1.0	Notes
Date 2	1.1	Notes

Contents

1 Revision History **i**

2 Introduction **1**

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

4 Module Hierarchy **2**

5 Connection Between Requirements and Design **4**

6 Module Decomposition **4**
- 6.1 Hardware Hiding Modules (M1) . . . 5
- 6.2 Behaviour-Hiding Module . . . 5
  - 6.2.1 Import csv Module (M2) . . . 5
  - 6.2.2 Import dm3 Module (M3) . . . 5
  - 6.2.3 Import h5 Module (M4) . . . 6
  - 6.2.4 Import dm3 Module (M5) . . . 6
  - 6.2.5 Export csv Module (M6) . . . 6
  - 6.2.6 Export h5 Module (M7) . . . 6
  - 6.2.7 Export png Module (M8) . . . 6
  - 6.2.8 Export rpl Module (M9) . . . 7
  - 6.2.9 Data Processing: Richardson-Lucy Deconvolution (M10) . . . 7
- 6.3 Software Decision Module . . . 7
  - 6.3.1 Etc. . . . 7

7 Traceability Matrix **7**

8 Use Hierarchy Between Modules **8**

List of Tables

1 Module Hierarchy . . . 4
2 Trace Between Requirements and Modules . . . 8
3 Trace Between Anticipated Changes and Modules . . . 8

List of Figures

1 Use Hierarchy . . . 9
2 Use Hierarchy I/O . . . 9

3 Use Hierarchy Data Processing 10
4 Use Hierarchy Data Extraction 10
5 Use Hierarchy Display 11
6 Use Hierarchy Data 11

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 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 data.

AC4: The algorithms used for processing data.

AC5: The type of input data.

AC6: The plotting library used.

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.

...

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: Import .csv Module

M3: Import .dm3 Module

M4: Import .h5 Module

M5: Import .rpl Module

M6: Export .csv Module

M7: Export .h5 Module

M8: Export .png Module

M9: Export .rpl Module

M10: Data Processing Richardson-Lucy Deconvolution Module

M11: Data Processing Normalization Module

M12: Data Processing Gain Correction Module

M13: Data Processing Background Correction Module

M14: Data Extraction 1D Slice Module

M15: Data Extraction 2D Mask Module

M16: Data Extraction 3D Mask Module

M17: Display 1D Spectrum Module

M18: Display 2D Image Module

M19: Display 3D Spectrum Image Module

M20: Data 1D Spectrum Module

M21: Data 2D Image Module

M22: Data 3D Spectrum Image Module

M23: Array Data Structure

M24: Plotting Structure

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 structure	

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. 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.

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 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 appropriate data module.

Implemented By: SpectrumImageAnalysisPy

6.2.2 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 appropriate data module. Will assign calibrations based on the metadata in the file.

Implemented By: SpectrumImageAnalysisPy

6.2.3 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 appropriate data module. Will assign calibrations based on the metadata in the file.

Implemented By: SpectrumImageAnalysisPy

6.2.4 Import dm3 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 appropriate data module. Will assign calibrations based on the metadata in the file.

Implemented By: SpectrumImageAnalysisPy

6.2.5 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.

Implemented By: SpectrumImageAnalysisPy

6.2.6 Export h5 Module (M7)

Secrets: Export data to h5 file.

Services: Takes spectrum or spectrum image data, formats it as appropriate, and exports the spectrum range, calibrations and metadata, and the spectrum image data to an h5 file.

Implemented By: SpectrumImageAnalysisPy

6.2.7 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.

Implemented By: SpectrumImageAnalysisPy

6.2.8 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.

Implemented By: SpectrumImageAnalysisPy

6.2.9 Data Processing

Secrets: Container for data processing modules.

Services: Performs data processing functions, as defined by Level 3 modules.

Implemented By: –

6.2.10 Data Processing: Richardson-Lucy Deconvolution (M10)

Secrets: Richardson-Lucy deconvolution algorithm.

Services: Performs Richardson-Lucy deconvolution on the desired spectrum or spectrum image input, using the number of iterations desired by the user and returns a deconvolved spectrum or spectrum image.

Implemented By: SpectrumImageAnalysisPy

6.2.11 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.

Implemented By: SpectrumImageAnalysisPy

6.2.12 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.

Implemented By: SpectrumImageAnalysisPy

6.2.13 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.

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 Etc.

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**
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
AC6	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

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 **??** 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.

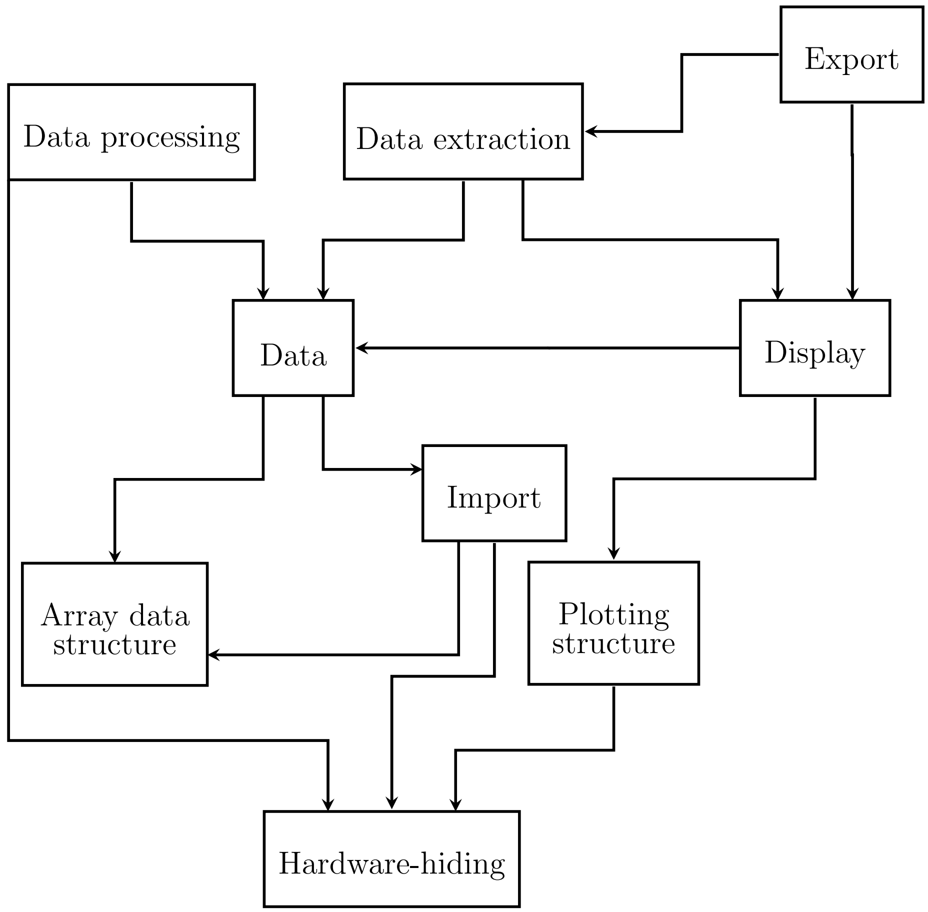

Figure 1: Use Hierarchy

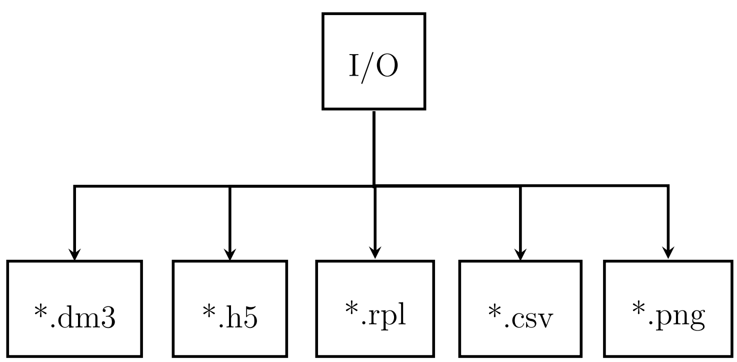

Figure 2: Use Hierarchy I/O

References

David L. Parnas. On the criteria to be used in decomposing systems into modules. *Comm. ACM*, 15(2):1053–1058, December 1972.

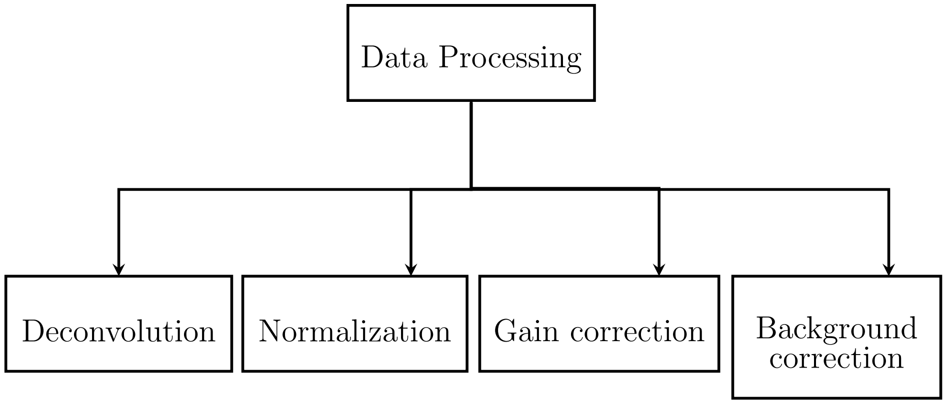

Figure 3: Use Hierarchy Data Processing

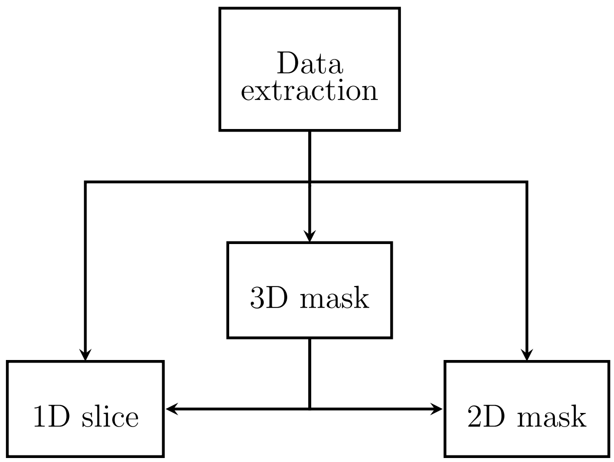

Figure 4: Use Hierarchy Data Extraction

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.

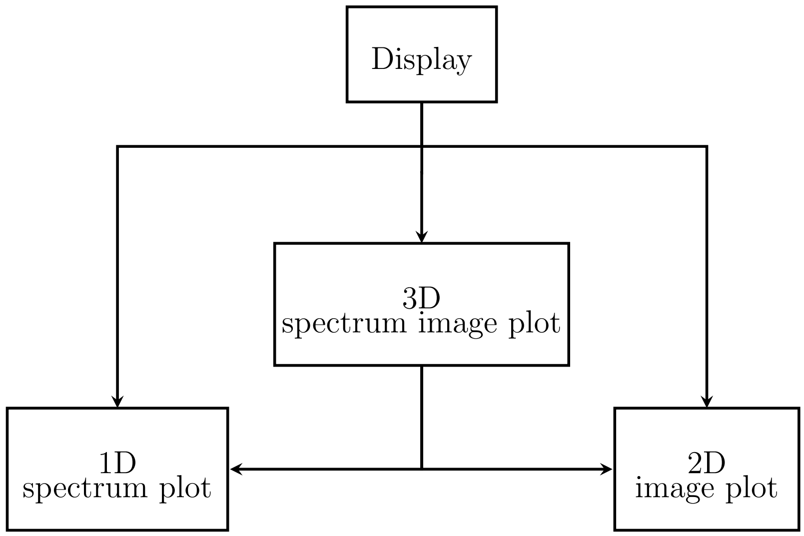

Figure 5: Use Hierarchy Display

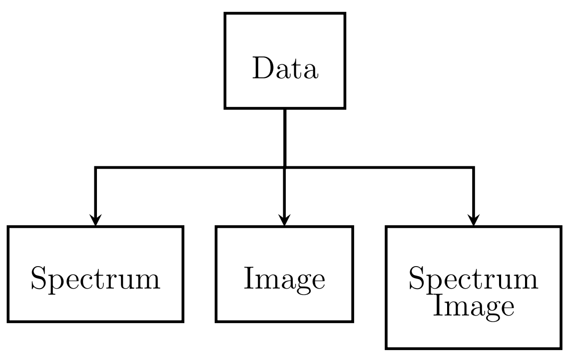

Figure 6: Use Hierarchy Data