Module Guide (MG)
STEM Moiré GPA

Alexandre Pofelski
macid: pofelska
github: slimpotatoes

November 5, 2017

1 Revision History

Table 1: **Revision History**

Date	**Version**	**Notes**
5/11/2017	1.0	First Draft

Contents

1 Revision History **i**

2 Introduction **1**

3 Anticipated and Unlikely Changes **1**
- 3.1 Anticipated Changes . . . 1
- 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 (M 1) . . . 4
- 6.2 Behaviour-Hiding Module . . . 4
  - 6.2.1 STEM Moiré GPA Control Module (M 2) . . . 4
  - 6.2.2 STEM Moiré GPA GUI (M 3) . . . 4
  - 6.2.3 Input Module (M 4) . . . 5
  - 6.2.4 SMH simulation Module (M 5) . . . 5
  - 6.2.5 GPA Module (M 6) . . . 5
  - 6.2.6 Mask Module (M 7) . . . 5
  - 6.2.7 Unstrained region Module (M 8) . . . 5
  - 6.2.8 Conversion Module (M 9) . . . 6
  - 6.2.9 2D strain tensor Module (M 10) . . . 6
- 6.3 Software Decision Module . . . 6
  - 6.3.1 Fourier Transform Module (M 11) . . . 6
  - 6.3.2 Gradient Module (M 12) . . . 6
  - 6.3.3 Least Square Fitting Method Module (M 13) . . . 6
  - 6.3.4 Phase Module (M 14) . . . 7
  - 6.3.5 Data Structure Module (M 15) . . . 7
  - 6.3.6 Object Structure Module (M 16) . . . 7
  - 6.3.7 Generic GUI/Plot Module (M 17) . . . 7

7 Traceability Matrix **7**

8 Use Hierarchy Between Modules **8**

List of Tables

1 **Revision History** . . . i

2 Module Hierarchy . . . 3

3 Trace Between Requirements and Modules 8
4 Trace Between Anticipated Changes and Modules 8

List of Figures

1 Use hierarchy among modules 9

2 Introduction

The module guide document is providing to the reader the decomposition of STEM Moiré GPA into smaller pieces to help the implementation phase. The decomposition is based on the principle of information hiding [1] which interest relies on its flexibility to design change. The module design follows the rules layed out by [2], 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.

The module guide (MG) follows the Software Requirements Specification (SRS) available in the documentation tree. Terminologies, symbols and acronyms used in the document are described in the SRS and TestPlan documents. 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 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.

AC 1 Hardware on which is STEM Moiré GPA running

AC 2 $I_{SMH_{\text{exp}}}, I_{C_{\text{ref}}}$ inputs file format

AC 3 Small strain and small gradient of strain assumption

AC 4 SMH simulation from a reference algorithm

AC 5 GPA algorithm

AC 6 Mask function for frequency isolation (in GPA)

AC 7 Fitting algorithm to get unstrained reference

AC 8 Output format

AC 9 Internal Data strucutre

AC 10 Internal Object strucutre

3.2 Unlikely Changes

The unlikely changes are design decisions fixing some aspects of STEM Moiré GPA. The lack of modularity is compensated by a simplification in the software design. If any of the elements are modified, important changes would be expected in the design. Therefore the following decisions are expected to stay unchanged.

UC 1 SMH data type (2D arrays)

UC 2 Perfect 2D periodic scanning grid assumption

UC 3 Pixel data type (real number)

UC 4 Keyboard/mouse interface device with user

4 Module Hierarchy

This section provides an overview of the module design. Modules are summarized in a hierarchy decomposed by secrets in table 2. The modules listed below, which are leaves in the hierarchy tree, are the modules that will actually be implemented.

M 1 Hardware-Hiding Module

M 2 STEM Moiré GPA Control Module

M 3 STEM Moiré GPA GUI Module

M 4 User input Module

M 5 SMH simulation Module

M 6 GPA Module

M 7 Mask Module

M 8 Unstrained region Module

M 9 Conversion Module

M 10 2D strain tensor Module

M 11 Fourier Transform Module

M 12 Gradient Module

M 13 Least Square Fitting Method Module

M 14 Phase operation Module

M 15 Data Structure Module

M 16 Object Structure Module

M 17 Generic GUI/Plot Module

Level 1	**Level 2**
Hardware-Hiding Module	
Behaviour-Hiding Module	Input STEM Moiré GPA Control STEM Moiré GPA GUI User Input SMH simulation GPA Mask Unstrained region Conversion 2D strain tensor
Software Decision Module	Fourier Transform Least square fitting method Phase calculation Gradient Generic GUI/Plot Data structure Object structure

Table 2: 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 3.

6 Module Decomposition

Modules are decomposed according to the principle of "information hiding" proposed by [2]. 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 (M 1)

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

6.2.1 STEM Moiré GPA Control Module (M 2)

Secrets: Execution flow of STEM Moiré GPA.

Services: Calls the different modules in the appropriate order.

Implemented By: STEM Moiré GPA

6.2.2 STEM Moiré GPA GUI (M 3)

Secrets: Methods to interact with user to make run STEM Moiré GPA.

Services: Serves as interface between the user and the software through the hardware by outputting calculation results and collecting user information (user inputs) for the calculation modules.

Implemented By: STEM Moiré GPA

6.2.3 Input Module (M 4)

Secrets: The format and structure of the input data.

Services: Loads, verifies and stores the input data into the appropriate data and object structure.

Implemented By: STEM Moiré GPA

6.2.4 SMH simulation Module (M 5)

Secrets: Algorithm to simulate the SMH from the reference image $I_{C_{\mathtt{Ref}}}$.

Services: Simulates the SMH in Fourier space using the pixel size p of $I_{SMH_{\mathtt{exp}}}$ and the Fourier transform the image reference $I_{C_{\mathtt{ref}}}$.

Implemented By: STEM Moiré GPA

6.2.5 GPA Module (M 6)

Secrets: GPA algorithm.

Services: Calculates the variation of masked Moiré wave vector on each element of the array.

Implemented By: STEM Moiré GPA

6.2.6 Mask Module (M 7)

Secrets: Algorithm to isolate a portion of an image.

Services: Isolates and weights a sub area of an image by fixing the parameter M_j.

Implemented By: STEM Moiré GPA

6.2.7 Unstrained region Module (M 8)

Secrets: Algorithm to calculate the unstrained reference.

Services: Calculates the unstrained reference based on the user input U and the result of the fitting method.

Implemented By: STEM Moiré GPA

6.2.8 Conversion Module (M 9)

Secrets: Algorithm to convert a Moiré wave vector to a crystalline wave vector.

Services: Performs the affine vectorial transformation using the Moiré wave vectors and the sampling vectors as inputs.

Implemented By: STEM Moiré GPA

6.2.9 2D strain tensor Module (M 10)

Secrets: Algorithm to calculate the strain from distortion matrix.

Services: Calculates the strain tensor on each pixel of the array.

Implemented By: STEM Moiré GPA

6.3 Software Decision Module

6.3.1 Fourier Transform Module (M 11)

Secrets: Fourier transform algorithm.

Services: Transforms data into its Fourier transform.

Implemented By: Python library

6.3.2 Gradient Module (M 12)

Secrets: 2D Gradient algorithm.

Services: Performs the 2D derivative of function.

Implemented By: Python library

6.3.3 Least Square Fitting Method Module (M 13)

Secrets: Least Square Fitting algorithm.

Services: Fits a set of data into a 2D linear function using the Least squares method.

Implemented By: Python library

6.3.4 Phase Module (M 14)

Secrets: Algorithm to wrap/unwrap the phase.

Services: Wraps/unwraps the phase by adding/removing discontinuities in the interval [0,2π].

Implemented By: Python library

6.3.5 Data Structure Module (M 15)

Secrets: Data format for an image.

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

Implemented By: Python library

6.3.6 Object Structure Module (M 16)

Secrets: Format for the object carried by the control module.

Services: Provides object format usable by module called by the control module.

Implemented By: Python library

6.3.7 Generic GUI/Plot Module (M 17)

Secrets: Generic methods to interact with the user.

Services: Provides the generic interface methods such as buttons, windows, plotting, entry fields to interact with a user.

Implemented By: Python library

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
R 1	M 3, M 2, M 1
R 2	M 4
R 3	M 5, M 2
R 4	M 3, M 1
R 5	M 3, M 1
R 6	M 4
R 7	M 6, M 7, M 2
R 8	M 3, M 1
R 9	M 4
R 10	M 8, M 2
R 11	M 3, M 1
R 12	M 9, M 4, M 2
R 13	M 10
R 14	M 3, M 1

Table 3: Trace Between Requirements and Modules

AC	Modules
AC 1	M 1
AC 2	M 4
AC 3	M 10
AC 4	M 5
AC 5	M 6
AC 6	M 7
AC 7	M 13
AC 8	M 3
AC 9	M 15
AC 10	M 16

Table 4: 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 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.

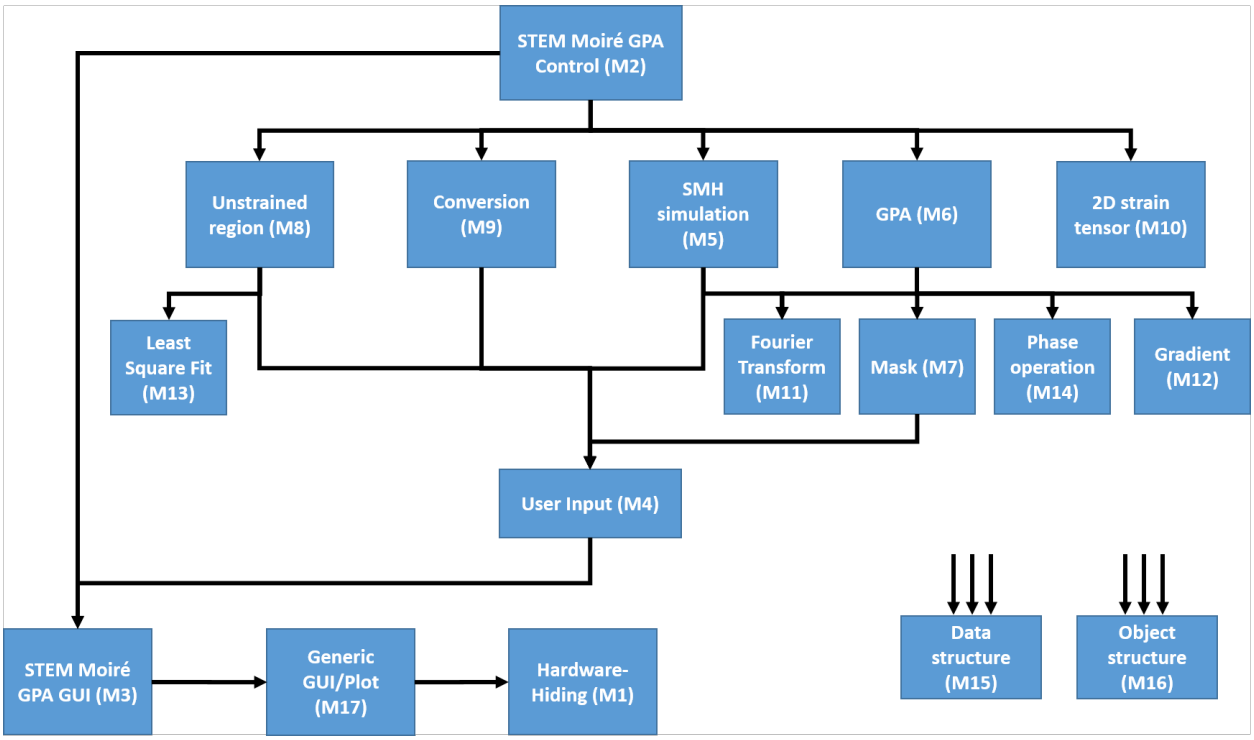

Figure 1: Use hierarchy among modules

References

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

[2] D. Parnas, P. Clement, and D. M. Weiss, "The modular structure of complex systems," in *International Conference on Software Engineering*, pp. 408–419, 1984.

[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.