Module Guide for ImgBeamer

Joachim de Fourestier

March 12, 2023

1 Revision History

Date	Version	Notes
2023/03/05	0.1.0	Creation
2023/03/10	0.1.1	Add anticipated changes
2023/03/12	0.1.2	Add unlikely changes
	0.1.3	List the modules

2 Reference Material

This section records information for easy reference.

2.1 Abbreviations and Acronyms

symbol	**description**
AC	Anticipated Change
DAG	Directed Acyclic Graph
DOM	Document Object Model
M	Module
MG	Module Guide
OS	Operating System
R	Requirement
SC	Scientific Computing
SRS	Software Requirements Specification
ImgBeamer	SEM image formation demo tool
UC	Unlikely Change

Contents

List of Tables

List of Figures

3 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 [4]. 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 Parnas et al. [4], as follows:

- System details that are likely to change independently should be the secrets of separate modules.
- Each data structure is implemented 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 [4]. 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 4 lists the anticipated and unlikely changes of the software requirements. Section 5 summarizes the module decomposition that was constructed according to the likely changes. Section 6 specifies the connections between the software requirements and the modules. Section 7 gives a detailed description of the modules. Section 8 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 9 describes the use relation between modules.

4 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 4.1, and unlikely changes are listed in Section 4.2.

4.1 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 input parameters.

AC4: The constraints on the input parameters.

AC5: The format of the final output data.

AC6: The constraints on the output results.

AC7: How the overall control of the calculations are orchestrated.

AC8: The display of scale in virtual, physical, or real world units such as nm.

AC9: The ability to simulate basic image noise (such as Gaussian or Poisson).

AC10: The ability to use different image quality metrics.

AC11: The implementation of compositing or blending operations (e.g., "Bit BLT").

AC12: The calculation of the average pixel or signal (intensity) value sampled by the beam.

AC13: The implementation of graphical user interface.

4.2 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: The software will not simulate electron beam physics, collision cascades, sample nature, topography, or other electron-sample interactions.

UC3: The beam sampling layout or raster pattern.

UC4: The software must be able to import a groud truth image.

UC5: The software must be able to export the resulting image.

UC6: The software must be able to display a live visualization to the user of the effects of changing the imaging parameters.

5 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: Ground Truth Image Input

M3: Imaging Parameters Input

M4: Spot Profile Control

M5: Image Export

M6: Image Information and Metrics Display

M7: Image Metrics Calculation

M8: Drawing Stage / Canvas Module

M9: Image Compositing

M10: Image Compositing

M11: Image Sampling

M12: DOM Manipulation

M13: Ground Truth Visualization

M14: Subregion Visualization

M15: Spot Content Visualization

M16: Spot Signal Visualization

M17: Spot Layout Visualization

M18: Sampled Subregion Visualization

M19: Resulting Subregion Visualization

M20: Resulting Image Visualization

Level 1	**Level 2**
Hardware-Hiding Module	
Behaviour-Hiding Module	Ground Truth Image Input Imaging Parameters Input Spot Profile Control Image Information and Metrics Display Image Export Ground Truth Visualization Subregion Visualization Spot Content Visualization Spot Signal Visualization Spot Layout Visualization Sampled Subregion Visualization Resulting Subregion Visualization Resulting Image Visualization
Software Decision Module	Image Compositing Image Rendering Image Sampling Drawing Stage / Canvas Module Image Metrics Calculation DOM Manipulation

Table 1: Module Hierarchy

6 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.

7 Module Decomposition

Modules are decomposed according to the principle of "information hiding" proposed by Parnas et al. [4]. 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. If the entry is *Web browser*, this means that the module is provided by the HTML 5 compliant web browser. If the entry is *CanvasAPI*, this means that the module is provided by the Canvas API of the HTML 5 living standard [5]. If the entry is *Konva*, this means that the module is provided by the Konva.js HTML5 2d canvas javascript library [1]. *ImgBeamer* means the module will be implemented by the ImgBeamer 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.

7.1 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

7.2 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: –

7.2.1 Input Format Module (M??)

Secrets: The format and structure of the input data.

Services: Converts the input data into the data structure used by the input parameters module.

Implemented By: [Your Program Name Here]

Type of Module: [Record, Library, Abstract Object, or Abstract Data Type] [Information to include for leaf modules in the decomposition by secrets tree.]

7.2.2 Etc.

7.3 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: –

7.3.1 Etc.

8 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
AC2	M??
AC3	M??
AC4	M??
AC5	M??
AC6	M??
AC7	M??
AC8	M??
AC9	M??
AC10	M??
AC??	M??
AC12	M??
AC13	M??

Table 3: Trace Between Anticipated Changes and Modules

9 Use Hierarchy Between Modules

In this section, the uses hierarchy between modules is provided. Parnas [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] Anton Lavrenov. Konva.js - JavaScript 2d canvas library, December 2021. URL https://konvajs.org/index.html.

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

[3] 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.

[4] 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.

[5] W3C. HTML living standard, the canvas element, Mar 2023. URL https://html.spec.whatwg.org/multipage/canvas.html.