Module Guide for AutoVox

Team #10, Five of a Kind
Omar Abdelhamid
Daniel Maurer
Andrew Bovbel
Olivia Reich
Khalid Farag

January 21, 2026

1 Revision History

Date	Version	Notes
November 10, 2025	1.0	Initial draft by All
January 21, 2026	2.0	Revision 0 by Daniel and Omar

2 Reference Material

This section records information for easy reference.

2.1 Abbreviations and Acronyms

symbol	**description**
AC	Anticipated Change
API	Application Programming Interface
AutoVox	Explanation of program name
CAD	Computer-Aided Design
CSV	Comma Separated Values
DAG	Directed Acyclic Graph
M	Module
MG	Module Guide
OBJ	Object file format (3D model)
OS	Operating System
R	Requirement
SC	Scientific Computing
SRS	Software Requirements Specification
STL	Stereolithography file format
UI	User Interface
UC	Unlikely Change
XML	Extensible Markup Language
3D	Three Dimensional

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 (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 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 (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 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 format of the initial input file (CAD file formats). Initially, the system will support STL files, but anticipated changes could require supporting additional CAD formats used in the industry. The change in input format affects the parsing logic, error handling, and integration with visualization manager. Changes will require updates to file importers and related data validation processes.

AC2: The constraints on the input parameters such as the maximum number of voxels and the maximum number of layers that can be displayed. These limits are currently fixed at 13,996,800,000 voxels and 518,400 voxels, but may need to be configurable in the future. Changes to these constraints will require updates to the input validation and parsing logic.

AC3: The dimensions of the voxel grid. The dimensions of the voxel grid are currently fixed at $300 \times 300\,\mu$m (XY) and $110\,\mu$m (Z), but may need to be configurable in the future as the user may want to use a different voxel size. Changes to these constraints will require updates to the voxelization algorithm and the layer navigation logic.

AC4: The algorithm used to convert the mesh CAD file into a voxel grid. Anticipated changes will require updates to the voxelization algorithm to improve the accuracy, performance, or new features.

AC5: The representation and format of magnetization direction data placed on the voxels, either by vector representation or coordinate system. This could involve changing the data structure (e.g., from Cartesian coordinates to Euler angles), supporting both absolute and relative directions, or accommodating multiple coordinate systems. Changes will require updates to the magnetization assignment logic.

AC6: The implementation of selecting multiple voxels. Currently multi-selection is supported by selecting each voxel individually or an entire layer at once. Changes will require updates to the selection logic considering a more flexible selection method such lasso selection or rectangle selection.

AC7: The exported file format for the custom printer software is currently CSV. The file must contain the voxel location, layer, and magnetization direction and material ID data. Anticipated changes could require supporting other formats (such as JSON, XML, or other file formats). Changes will require updates to the export logic.

AC8: The implementation of the data structures used to store and manage voxel grids and voxel properties. This could involve using different data structures such as a tree structure or a graph structure optimizing the performance metrics of the system, metrics such as memory usage, search speed, or parallel processing. Anticipating such changes allows for future performance and scalability enhancements. Changes will require updates to the data structures.

AC9: The layout and interaction methods of the user interface, such as menu structure, keyboard shortcuts, and user feedback mechanisms. This anticipated change helps accommodate different user workflows, accessibility requirements, or even support for new input devices. Changes will require updates to the UI presentation or logic.

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. The system is designed with the expectation that it operates with a limited and specific set of input and output devices. Input is handled through files (such as CAD input files or configuration files) and/or keyboard entry (for command-line or interactive parameter input). Output is provided via files (for export and logging), memory (for internal data storage and communication between modules), and/or a graphical screen (for visualization and user interface feedback).

UC2: The overall system architecture. The software follows a rigid four-manager structure that divides system functionality into Import, Visualization, Editing, and Export managers. Each manager is responsible for a core facet of application operation, and this separation is central to the modular design.

UC3: The ordering of layers from bottom to top (Z-axis ordering). Layers within the voxel grid are organized along the Z-axis, proceeding from the lowest layer (the base) to the highest (the top). This convention aligns with common additive manufacturing and 3D modeling practices.

UC4: The fundamental user interaction paradigm. The system targets a desktop usage scenario, using mouse and keyboard as primary input methods. This supports rapid interaction, direct manipulation of interface elements, and fine control over editing tasks.

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: Input Interpreter Module

M2: Voxel Slicing Module

M3: Display Partitioning Module

M4: Project Manager Module

M5: Interaction Controller Module

M6: Visualization State Manager Module

M7: Model Manager Module

M8: History Manager Module

M9: Voxel Tracking Module

M10: Highlight Manager Module

M11: Export Validation Module

M12: Export Manager Module

M13: Error Diagnostic Handler Module

M14: Model Structure Module

M15: Graphics Adapter Module

M16: Export Structure Module

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 1.

Table 1: Module Hierarchy

Level 1	**Level 2**
Hardware-Hiding Module	None
Behaviour-Hiding Module	Input Interpreter Module
	Voxel Slicing Module
	Display Partitioning Module
	Project Manager Module
	Interaction Controller Module
	Visualization State Manager Module
	Model Manager Module
	History Manager Module
	Voxel Tracking Module
	Highlight Manager Module
	Export Validation Module
	Export Manager Module
	Error Diagnostic Handler Module
Software Decision Module	Model Structure Module
	Graphics Adapter Module
	Export Structure Module

7 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. *AutoVox* means the module will be implemented by the AutoVox 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 (None)

This system has no hardware components.

7.2 Behaviour-Hiding Module

7.2.1 Input Interpreter Module (M1)

Secrets: How to extract, parse, and normalize data from any compatible input file into the model's internal format. How to identify file type, and how to parse and interpret 3D geometry.

Services: Provides a unified import interface for all supported input formats. When the user selects a CAD file to load, this module determines the file type and converts it into a consistent intermediate representation. For STL files, it extracts geometric mesh data (facets, vertices, normals) and scales it to the target voxel grid resolution.

Implemented By: AutoVox

Type of Module: Library: a reusable parsing and normalization component that handles all input data extraction for new projects.

7.2.2 Voxel Slicing Module (M2)

Secrets: How to convert the input geometry into a set of voxels. How to slice along all axes and optimize voxelization resolution.

Services: Converts geometric mesh into voxel representation. Generates discrete voxels by slicing through the 3D model at fixed intervals.

Implemented By: AutoVox

Type of Module: Library: a computational module responsible for transforming geometry into voxelized form.

7.2.3 Display Partitioning Module (M3)

Secrets: How to determine the boundaries that define model partitions. How to define the relationship between display segments. How to represent partition boundaries. How to minimize the number of partitions created. How to ensure that display partitions maintain balanced spatial proportions. How to validate that a partition of voxels is viable to display within the UI.

Services: Provide functionality for partitioning the model into distinct display segments defined by specific voxel groupings. Manages the relationships and alignment between display segments.

Implemented By: AutoVox

Type of Module: Library: a reusable set of functions responsible for defining and partitioning sets of voxels into display segments.

7.2.4 Project Manager Module (M4)

Secrets: How to create and initialize a new project workspace. How to establish the intitial project structure, metadata, and default configuration settings. How to allocate and manage project resources and state. How to save and load projects and how frequently to autosave. How multiple workspaces are managed.

Services: Creates new projects with appropriate initialization. Manages loading projects, and the autonomous and manual saving of projects. Manages swapping between workspaces.

Implemented By: AutoVox

Type of Module: Abstract Object: a component responsible for project initialization and workspace management.

7.2.5 Interaction Controller Module (M5)

Secrets: How events are internally handled. How to differentiate between similar events. How to define the rules that associate events with actions to take. How to handle events where user intent is inconclusive. How to determine when multiple events should be interpreted as a single action. How to apply context from UI when interpreting events.

Services: Maps events from interaction with UI tosystem actions that reflect user intent.

Implemented By: AutoVox

Type of Module: Abstract Object: a component responsible for internal event handling and action routing that internally must track the current view displayed to the user for context.

7.2.6 Visualization State Manager Module (M6)

Secrets: How visualization-related state is represented and maintained independently of model modification, user interaction, and persistence mechanisms. How changes to the underlying model are reflected as visualization state transitions without exposing rendering or interaction logic.

Services: Maintains the current visualization state of the system, including active views and highlight states. Coordinates visualization state updates by aggregating state change descriptors from upstream modules and forwarding structured update requests to the Graphics Adapter.

Implemented By: AutoVox

Type of Module: Abstract Object: a component responsible for maintaining visualization state and coordinating visualization updates.

7.2.7 Model Manager Module (M7)

Secrets: How model state is represented and mutated, including voxel updates and material and magnetization assignments. How model state is persisted through explicit save operations of the model file.

Services: Manages voxel-based model state by providing operations to add, remove, and modify voxels and update material and magnetization properties. Provides explicit save operations for persisting model state. Maintains model-wide consistency and exposes model state changes to dependent modules without coordinating visualization or user interaction.

Implemented By: AutoVox

Type of Module: Abstract Object: a component responsible for model state management and persistence.

7.2.8 History Manager Module (M8)

Secrets: How changes to the voxel grid and project states are tracked and stored, including what the previous (past) version was and what the current version is after modification. How storage and retrieval of specific data from past history are handled to enable undo and redo functionality. How record-keeping is managed to ensure reliable and efficient state transitions.

Services: Tracks and stores changes to the voxel grid and project states. Enables undo and redo functionality by restoring previous states. Manages storage and retrieval of specific data from past history.

Implemented By: AutoVox

Type of Module: Abstract Object: a computational component for tracking and storing changes to the voxel grid and project states.

7.2.9 Voxel Tracking Module (M9)

Secrets: How to locate and track voxels that satisfy particular property criteria, such as selection state, material type, or user-defined rules. How to interpret the voxel grid data structure to efficiently identify and access relevant voxels without exposing model mutation or interaction logic.

Services: Interprets the voxel data structure to find and return voxels meeting specified property criteria (e.g., all voxels with a given material, within a spatial region, or within a given layer). Enables other modules (such as Highlight Manager) to query voxel sets based on their properties.

Implemented By: AutoVox

Type of Module: Abstract Object: a computational component that provides read-only querying and tracking of voxels with specific properties within the voxel grid.

7.2.10 Highlight Manager Module (M10)

Secrets: How to define the rules that associate voxel semantics with a highlight colour. How highlight colours are internally represented. How conflicts are resolved when a voxel qualifies for multiple highlight colours simultaneously. How colour palette, which encapsulates all highlight colours, is chosen. How accessibility requirements influence the selection of highlight colours.

Services: Maps relationship between highlight colour and semantic meaning of highlighted voxels.

Implemented By: AutoVox

Type of Module: Abstract Object

7.2.11 Export Validation Module (M11)

Secrets: How to validate whether a project metadata file is suitable for export. How to check export file format requirements, verify that all voxels have required material and magnetization properties, and ensure data integrity constraints are satisfied. How to determine export readiness based on validation rules.

Services: Validates export readiness by checking export file format requirements, verifying completeness of voxel properties (material and magnetization), and ensuring all export constraints are met. Provides validation results and error reporting for export issues.

Implemented By: AutoVox

Type of Module: Library: A reusable component providing validation functionality for export operations.

7.2.12 Export Manager Module (M12)

Secrets: How to export a project with the defined data structure format. How to transform internal model data into the export file format. How to serialize project data including voxel grids, metadata, material properties, and magnetization information according to export specifications.

Services: Exports projects to external file formats with the defined data structure. Handles the conversion of internal model representations to export-compatible formats, ensuring data integrity and completeness during the export process.

Implemented By: AutoVox

Type of Module: Abstract Object: A component responsible for coordinating the export process and data transformation.

7.2.13 Error Diagnostic Handler Module (M13)

Secrets: How to detect and communicate errors related to unresponsiveness of the model, graphics update failures, and model file issues. How to classify error types and determine error sources. How to handle error recovery and logging mechanisms.

Services: Detects and diagnoses errors occurring during model updates, graphics rendering, or file operations. Provides error classification and logging.

Implemented By: AutoVox

Type of Module: Library: A reusable component providing error detection, diagnosis, and handling functionality.

7.3 Software Decision Module

7.3.1 Model Structure Module (M14)

Secrets: How to define and implement the internal data structure for representing a 3D voxel grid, including how layers are organized and indexed, and how voxel metadata (such as material and magnetization) is stored and accessed. How to ensure efficient access, retrieval, and consistency of voxel properties within the grid.

Services: Defines the data structure for 3D voxel grid representation with layer organization and voxel metadata. Specifies how voxel grids are stored as three-dimensional arrays, how layers group voxels by Z-coordinate, and how voxel properties including material assignments and magnetization vectors are managed.

Implemented By: AutoVox

Type of Module: Abstract Data Type: specifies and maintains the structure of 3D voxel grid data with layer organization and voxel metadata.

7.3.2 Graphics Adapter Module (M15)

Secrets: How to manage the persistence of project and model data, including how to store, index, and access the data. How to handle the specific file-system or database technology (e.g., CSV files) and its connection management. How to handle caching and concurrency handling mechanisms.

Services: Provides read and write access between the application and storage layer. Saves serialized models and retrieves them when requested. Ensures data integrity and version consistency using the Serialization Manager (M5).

Implemented By: AutoVox

Type of Module: Library: a component responsible for managing the persistence of project and model data.

7.3.3 Export Structure Module (M16)

Secrets: How to define and implement the internal data structure for representing a exported file, including field ordering, data types, and encoding format. How to ensure consistency between internal and external representations.

Services: Defines the data schema used during export. Specifies how voxel data, layer information, and material/magnetization fields are organized.

Implemented By: AutoVox

Type of Module: Abstract Data Type: specifies and maintains the structure of exported model data.

8 Traceability Matrix

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

Req.	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11	M12	M13	M14	M15	M16
F211	■	□	□	■	□	□	□	□	□	□	□	□	□	□	□	□
F212	■	□	□	■	□	□	□	□	□	□	□	□	□	□	□	□
F213	■	■	□	□	■	□	□	□	□	□	□	□	□	□	□	□
F214	■	□	□	□	■	□	□	□	□	□	□	□	□	□	□	□
F215	■	■	□	□	□	□	□	□	□	□	□	□	□	□	□	□
F216	□	□	■	□	□	□	□	□	□	□	□	□	□	□	□	□
F221	□	□	□	□	□	■	□	□	□	□	□	□	□	■	■	□
F222	□	□	■	□	■	■	□	□	□	□	□	□	□	■	□	□
F223	□	□	□	□	■	■	□	□	□	□	□	□	□	■	■	□
F224	□	□	□	□	■	■	□	□	□	□	□	□	□	■	■	□
F225	□	□	□	□	■	■	□	□	■	■	□	□	□	■	■	□
F226	□	□	□	□	□	■	□	□	■	■	□	□	□	■	■	□
F227	□	□	□	□	□	■	□	□	■	■	□	□	□	■	■	□
F231	□	□	□	□	■	□	■	□	□	□	□	□	□	■	□	□
F232	□	□	□	□	■	□	■	□	□	■	□	□	□	□	□	□
F233	□	□	□	□	■	□	■	□	□	□	□	□	□	■	□	□
F234	□	□	□	□	■	□	■	□	□	■	□	□	□	□	□	□
F235	□	□	□	□	■	□	■	■	■	□	□	□	□	■	□	□
F236	□	□	□	□	□	□	□	■	□	□	□	□	■	□	□	□
F237	□	□	□	□	■	□	■	■	□	□	□	□	□	□	□	□
F238	□	□	□	□	■	□	■	□	□	■	□	□	□	■	□	□
F239	□	□	□	□	■	□	■	□	□	□	□	□	□	■	□	□
F2310	□	□	□	□	■	□	■	□	□	□	□	□	□	■	□	□
F2311	□	□	□	■	□	□	□	□	□	□	□	□	□	□	□	□
F241	□	□	□	□	□	□	□	□	■	□	■	□	□	■	□	□
F242	□	□	□	□	□	□	□	□	□	□	□	■	□	■	□	■
F243	□	□	□	□	□	□	□	□	□	□	□	■	□	■	□	□
F244	□	□	□	□	□	□	□	□	□	□	□	■	■	□	□	■
F245	□	□	□	□	□	□	□	□	□	□	□	■	□	■	□	■

Table 2: Trace Between Functional Requirements and Modules

Req.	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11	M12	M13	M14	M15	M16
NF211	■	■	□	■	□	□	□	□	□	□	□	□	■	□	□	□
NF212	■	■	□	□	□	□	□	□	□	□	□	□	■	■	□	□
NF221	□	□	□	□	□	■	■	□	□	□	□	□	■	□	■	□
NF222	□	□	■	□	□	□	□	□	■	□	□	□	■	□	■	□
NF223	□	□	□	□	□	□	□	□	■	■	□	□	□	□	□	□
NF231	□	□	□	□	□	■	□	□	□	□	□	□	□	□	□	□
NF232	□	□	□	□	□	□	■	□	□	□	□	□	■	■	□	□
NF241	□	□	□	□	□	□	□	□	□	□	■	■	■	□	□	□
NF242	□	□	□	□	□	□	□	□	□	□	□	■	■	■	□	■

Table 3: Trace Between Non-Functional Requirements and Modules

Req.	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11	M12	M13	M14	M15	M16
AC1	■	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□
AC2	■	□	■	□	□	□	□	□	□	□	□	□	□	■	□	□
AC3	□	■	■	□	□	□	□	□	□	□	□	□	□	■	□	□
AC4	■	■	□	□	□	□	□	□	□	□	□	□	□	□	□	□
AC5	□	□	□	□	■	□	■	□	□	□	□	□	□	■	□	□
AC6	□	□	□	□	■	□	□	□	■	□	□	□	□	□	■	□
AC7	□	□	□	□	□	□	□	□	□	□	□	■	□	□	□	■
AC8	□	□	□	□	□	□	□	□	□	□	□	□	□	■	□	□
AC9	□	□	□	□	■	■	□	□	□	□	□	□	□	□	■	□

Table 4: Trace Between Anticipated Changes and Modules

9 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.

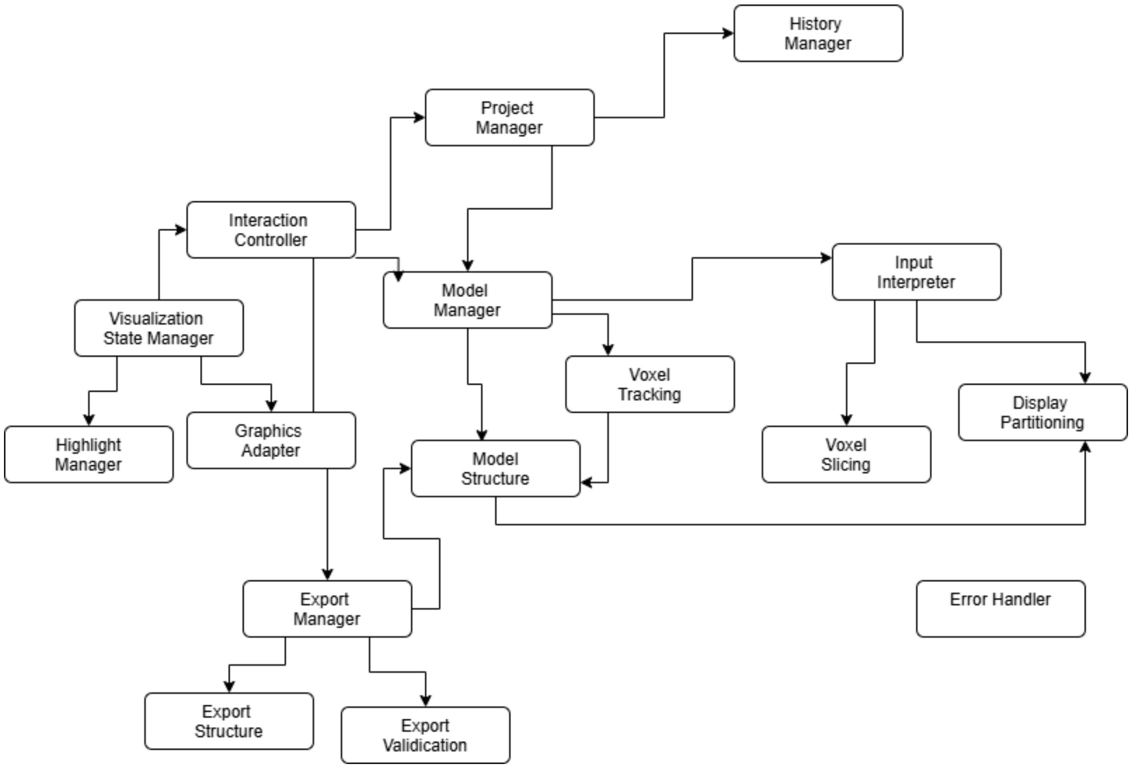

Figure 1: Use hierarchy among modules

Module Assignment and Implementation Timeline

Table 5 summarizes the implementation responsibility and planned completion timeline for each module in the system. Each module identifier is hyperlinked to its corresponding detailed module specification to support traceability between the software architecture and detailed design.

Module	**Assignee(s)**	**Completion Date**
M1: Input Interpreter	Andrew, Daniel	24 Nov 2025
M2: Voxel Slicing	Andrew, Daniel	24 Nov 2025
M3: Display Partitioning	Omar, Olivia	22 Jan 2026
M4: Project Manager	Khalid, Andrew	24 Jan 2026
M5: Interaction Controller	Andrew, Daniel	24 Nov 2025
M6: Visualization State Manager	Andrew, Daniel	24 Nov 2025
M7: Model Manager	Andrew, Daniel	26 Jan 2026
M8: History Manager	Khalid, Olivia	27 Jan 2026
M9: Voxel Tracking	Omar, Daniel	28 Jan 2026
M10: Highlight Manager	Omar, Olivia	29 Jan 2026
M11: Export Validation	Omar, Khalid	30 Jan 2026
M12: Export Manager	Khalid, Olivia	31 Jan 2026
M13: Error Diagnostic Handler	Andrew, Khalid	1 Feb 2026
M14: Model Structure	Omar, Daniel	25 Jan 2026
M15: Graphics Adapter	Olivia, Khalid	2 Feb 2026
M16: Export Structure	Andrew, Daniel	24 Nov 2025

Table 5: Module assignment and implementation timeline

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.