Module Guide for Software Engineering

Team 21, Alkalytics
Sumanya Gulati
Kate Min
Jennifer Ye
Jason Tran

January 16, 2025

1 Revision History

Date	Version	Notes
11 Janury 2025	1.0	Added initial content for Rev 0.

Note: Please note that our team has adapted and extended this Module Guide document to include the contents of an MIS or other such document. For this reason, only one design document has been submitted.

2 Reference Material

This section records information for easy reference.

2.1 Abbreviations and Acronyms

symbol	**description**
AC	Anticipated Change
API	Application Programming Interface
CSV	Comma Separated Values
DAG	Directed Acyclic Graph
JSON	JavaScript Object Notation
KPI	Key Performance Indicator
M	Module
MG	Module Guide
MIS	Module Interface Specification
OS	Operating System
R	Requirement
SC	Scientific Computing
SRS	Software Requirements Specification
Software Engineering	Explanation of program name
UC	Unlikely Change
UI	User Interface
XML	Extensible Markup Language

2.2 Notation

Contents

List of Tables

List of Figures

3 Introduction

3.1 Summary

Alkalytics is a project designed to provide a scalable data management and analysis solution for ocean alkalinity research, in particular by streamlining the data organization, querying, and visualization processes. Through its various modules, the primary goal of the system is to offer a comprehensive solution for data ingestion, processing and reporting while maintaining adaptability to future changes.

Each functional component is developed as an independent module to encapsulate specific responsibilities, minimize dependencies, and promote information hiding. This modular approach, advocated for widely in the software sector, not only simplifies development and testing but also allows the system to accommodate evolving user requirements and technology upgrades.

3.2 Purpose

This Module Guide (MG) has been written to serve as a roadmap for the Alkalytics system, detailing its structure, functionality, and the relationships between its components. It provides clarity on how the system meets the requirements outlined in the Software Requirements Specification (SRS) and supports the following stakeholders:

- **New Developers**: To understand the modular architecture and ensure consistent implementation.
- **Maintainers**: To efficiently identify, update, or rewrite modules as needed.
- **Designers**: To validate the system's feasibility, flexibility, and alignment with project goals.

4 Anticipated and Unlikely Changes

This section identifies potential changes to the system and classifies them into two categories: anticipated changes (AC) as listed in section 4.1 and unlikely changes (UC) as listed in section 4.2. AC represent decisions that have been encapsulated within specific modules to minimize the impact of modifications while UC are those that, while possible, are fixed at the system architecture stage to reduce complexity.

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. These changes are encapsulated within specific modules to ensure the system's adaptability.

AC1: Hardware Configuration - The software may need to run on a different hardware platform like a server or on a cloud solution. Changes in hardware specifications will primarily affect the 1 Module, isolating their imoact.

AC2: Data Processing Algorithms - Changes in analytical techniques along with advances in the machine learning space would introduce the need for new statistical models. These changes would be encapsulated in the Data Processing Module.

AC3: User Interface (UI) Design - Changes in analytical techniques along with advances in the machine learning space would introduce the need for new statistical models. These changes would be encapsulated in the Data Processing Module.

AC4: Input Data Formats - Currently, the system is expected to process data from Comma Separated Values (CSV) files only. In the future, however, modifications may have to be made to accommodate different file formats (such as JavaScript Object Notation (JSON), Extensible Markup Language (XML) etc) which will be handled by the Data Ingestion Module without impacting other parts of the system.

AC5: Data Source Integration - New data sources such as third-party application programming interfaces (APIs), Internet of Things (IoT) devices may have to be added in the future. The Data Integration Module will have to be redesigned to handle the integration of these new sources.

AC6: Scaling Data Volume - As the number of experiments increases, the system may need to handle increasing data volumes as usage grows. This is addressed by the Data Storage Module which has been designed to support database scalability strategies.

AC7: User Roles and Permissions - Future requirements may demand the addition of new user roles or changes to existing permissions. The Administration Module is designed to encapsulate these changes.

AC8: Input Schema - With an increase in the number of diverse experiments, the schema for the data inputs may have to changed to support the addition or removal of new parameters. This is handled by the Data Ingestion Module.

AC9: Notification Rules - The conditions of triggering alerts or notifications may evolve, including but not limited to additional thresholds or new types of anomalies. These are handled by the Notifications Module without affecting other parts of the system.

AC10: Analytical Metrics - New metrics or Key Performance Indicators (KPIs) might be requested by stakeholders. This would involve adapting requirements by introducing new calculations or processing pipelines by modifying the Data Processing Module.

4.2 Unlikely Changes

Unlikely changes are those that are fixed early in the design to simplify the system and reduce complexity. These changes, if necessary, would have a significant impact on multiple modules.

UC1: Input/Output Devices - The system is designed to support file-based inputs. Changes can include additional input and/or output methods such as direct hardware interaction, would require substantial redesign across multiple modules.

UC2: Core System Architecture - The underlying architectural decisions, such as the use of modular decomposition and separation of concerns, are not expected to change. Altering these decisions would necessitate a complete overhaul of the system.

UC3: Communication Protocols - The communication methods between modules such as function calls, API interactions etc are fixed. Switching to a different communication protocol would impact the interfaces of all interacting modules.

UC4: Programming Language - The choice of programming languages is asssumed to be fixed for the project. A change would require rewriting most of the system.

UC5: Database Type - The choice of storage solution (relational versus NoSQL databases, for example) is assumed to remain fixed. Switching to a different type of database would require reworking the Data Storage Module and parts of the Data Processing Module.

5 Module Hierarchy

This section provides an overview of the module design for the Alkalytics system. The modules are summarized in a hierarchy that follows the principle of information hiding. Each module encapsulates specific secrets, ensuring changes are localized and do not affect unrelated parts of the system. These modules are summarized in a hierarchy decomposed by secrets in table 1. This hierarchy represented as a directed acyclic graph, shown in **??**, shows relationships betweem higher-level and lower-level modules, with the leaf modules representing those that will actually be implemented.

M1: Hardware-Hiding Module

M2: Behaviour-Hiding Module

M3: Interface Module

M4: Administration Module

M5: Data Acquisition Module

M6: Data Storage Module

M7: Data Retrieval Module

M8: Input Module

M9: Processing Module

M10: Output Module

M11: UI Design Module

M12: Visualization Module

M13: User Management Module

M14: Configuration Management Module

M15: Data Ingestion Module

M16: Data Validation Module

M17: Data Transformation Module

M18: Machine Learning Module

M19: Reporting Module

M20: Notification Module

Level 1	**Level 2**	**Level 3**
Hardware-Hiding Module		
	Data Acquisition Module	
	Data Storage Module	
	Data Retrieval Module	
Behaviour-Hiding Module	Input Module	Data Ingestion Module
		Data Validation Module
	Processing Module	Data Transformation Module
		Machine Learning Module
	Output Module	Reporting Module
		Notifications Module
Interface Module	UI Design Module	
	Visualization Module	
Administration Module	User Management Module	
	Configuration Management Module	

Table 1: Module Hierarchy

It must be noted that the blue nodes shown in figure 1 represent 'container' modules that are non-leaf modules and encompass other leaf nodes. The orange nodes represent leaf modules that will not be implemented for Revision 0 of Alkalytics and the green nodes represent leaf modules that will be implemented.

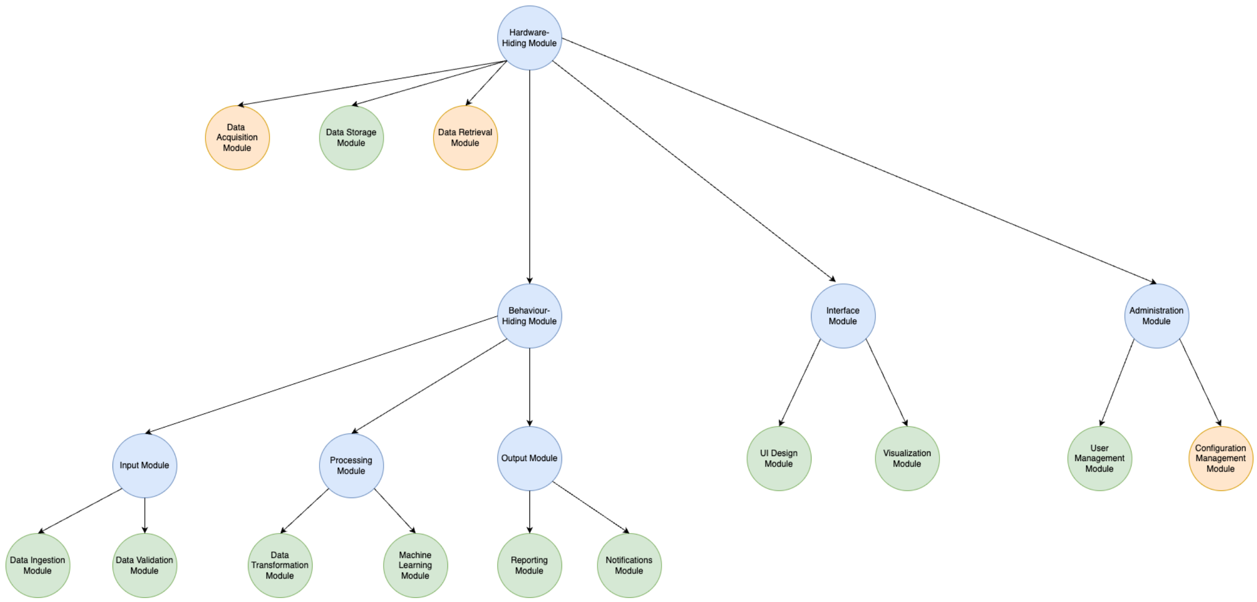

Figure 1: A DAG representing the implemented module hierarchy of Alkalytics.

6 Module Decomposition

Modules are decomposed according to the principle of "information hiding" proposed by Parnas et al. (1984). In this section, the *Secrets* field acts as 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. *Software Engineering* means the module will contain custom code and thus, will be implemented by the software engineering team.

Descriptions for non-leaf modules or the modules that will not be implemented for Revision 0 of Alkalytics have been provided below.

6.1 Data Storage Module (M6)

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

Rationale:

6.1.1 Uses

6.1.2 Syntax

Exported Constants:

Exported Access Programs:

6.1.3 Semantics

State Variables:

Environment Variables:

Assumptions:

Access Routine Semantics:

Local Function:

6.2 Data Ingestion Module (M15)

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

Rationale:

6.2.1 Uses

6.2.2 Syntax

Exported Constants:

Exported Access Programs:

6.2.3 Semantics

State Variables:

Environment Variables:

Assumptions:

Access Routine Semantics:

Local Function:

6.3 Data Validation Module (M16)

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

Rationale:

6.3.1 Uses

6.3.2 Syntax

Exported Constants:

Exported Access Programs:

6.3.3 Semantics

State Variables:

Environment Variables:

Assumptions:

Access Routine Semantics:

Local Function:

6.4 Data Transformation Module (M17)

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

Rationale:

6.4.1 Uses

6.4.2 Syntax

Exported Constants:

Exported Access Programs:

6.4.3 Semantics

State Variables:

Environment Variables:

Assumptions:

Access Routine Semantics:

Local Function:

6.5 Machine Learning Module (M18)

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

Rationale:

6.5.1 Uses

6.5.2 Syntax

Exported Constants:

Exported Access Programs:

6.5.3 Semantics

State Variables:

Environment Variables:

Assumptions:

Access Routine Semantics:

Local Function:

6.6 Reporting Module (M19)

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

Rationale:

6.6.1 Uses

6.6.2 Syntax

Exported Constants:

Exported Access Programs:

6.6.3 Semantics

State Variables:

Environment Variables:

Assumptions:

Access Routine Semantics:

Local Function:

6.7 Notifications Module (M20)

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

Rationale:

6.7.1 Uses

6.7.2 Syntax

Exported Constants:

Exported Access Programs:

6.7.3 Semantics

State Variables:

Environment Variables:

Assumptions:

Access Routine Semantics:

Local Function:

6.8 UI Design Module (M11)

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

Rationale:

6.8.1 Uses

6.8.2 Syntax

Exported Constants:

Exported Access Programs:

6.8.3 Semantics

State Variables:

Environment Variables:

Assumptions:

Access Routine Semantics:

Local Function:

6.9 Visualization Module (M12)

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

Rationale:

6.9.1 Uses

6.9.2 Syntax

Exported Constants:

Exported Access Programs:

6.9.3 Semantics

State Variables:

Environment Variables:

Assumptions:

Access Routine Semantics:

Local Function:

6.10 User Management Module (M13)

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

Rationale:

6.10.1 Uses

6.10.2 Syntax

Exported Constants:

Exported Access Programs:

6.10.3 Semantics

State Variables:

Environment Variables:

Assumptions:

Access Routine Semantics:

Local Function:

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
AC4	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 2 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.

[The uses relation is not a data flow diagram. In the code there will often be an import statement in module A when it directly uses module B. Module B provides the services that module A needs. The code for module A needs to be able to see these services (hence the import statement). Since the uses relation is transitive, there is a use relation without an import, but the arrows in the diagram typically correspond to the presence of import statement. —SS]

[If module A uses module B, the arrow is directed from A to B. —SS]

Figure 2: Use hierarchy among modules

9 User Interfaces

[Design of user interface for software and hardware. Attach an appendix if needed. Drawings, Sketches, Figma —SS]

10 Timeline

[Schedule of tasks and who is responsible —SS]

[You can point to GitHub if this information is included there —SS]

References

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.