Module Guide for Software Engineering

Team 17, Team RAdiAIdance
Allison Cook
Ibrahim Issa
Mohaansh Pranjal
Nathaniel Hu
Tushar Aggarwal

January 16, 2024

1 Revision History

Date	Version	Notes
01/11/2024	0.0	Initial Document
01/12/2024	0.1	Initial Draft of Anticipated and Unlikely Changes and Module Hierarchy sections; Added some terms to Abbreviations and Acronyms section
01/13/2024	0.2	Added Uses Hierarchy among modules diagram
01/14/2024	0.3	Filled out Traceability Matrices for Traceability Matrix section
01/15/2024	0.4	Made small edits to Table 1 in Module Hierarchy section and Table 4 in Traceability Matrix section; Initial Draft of Module Decomposition section completed

2 Reference Material

This section records information for easy reference.

2.1 Abbreviations and Acronyms

symbol	description
AC	Anticipated Change
AI/ML	Artificial Intelligence/Machine Learning
DAG	Directed Acyclic Graph
DICOM	Digital Imaging and Communications in Medicine; technical standard for digital storage/transmission of medical images and related information
GUI	Graphical User Interface
JPEG/JPG	Joint Photographic Experts Group; digital image compression standard, image format
M	Module
MG	Module Guide
MVC	Model-View-Controller Software Architecture
NLP	Natural Language Processing
OS	Operating System
R	Requirement
SC	Scientific Computing
SRS	Software Requirements Specification
Software Engineering	The Process of Designing and Developing Software; a reference to the software application described in this document
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 (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 specific hardware on which the software is running.

AC2: The specific selection of diseases that the AI/ML model will look for when performing the chest x-ray scan/read.

AC3: The output radiology report generated using NLP (progressing from report components to more complete report).

AC4: The various application pages/GUI (page layout changes/redesign, etc.).

AC5: The module for interfacing with the IT system(s) of the medical institution(s) using this application.

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 same (Python) library/libraries will be used for the AI/ML model.

UC3: The database/backend provider (Firebase) will be used for user authentication/management and application security.

UC4: The same backend/database provider (Firebase) will be used.

UC5: The input format of the chest x-ray data (DICOM) into the application.

UC6: The input format of the chest x-ray image (JPG or JPEG) into the AI/ML model for performing the chest x-ray scan/read.

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: AIModel Module

M2: ChestXRayRead Module

M3: ResultsGen Module

M4: NLPModel Module

M5: RepCompGen Module

M6: Backend Module

M7: DatabaseOps Module

M8: UserAuthMgmt Module

M9: MedInstInter Module

M10: AppController Module

M11: AppGUI Module

M12: Login Module

M13: PerfScan Module

M14: ViewResults Module

Level 1	**Level 2**
Hardware-Hiding Module	MedInstInter
Behaviour-Hiding Module	ChestXRayRead ResultsGen RepCompGen DatabaseOps UserAuthMgmt Login PerfScan ViewResults
Software Decision Module	AIModel NLPModel Backend AppController AppGUI

Table 1: Module Hierarchy

The modules listed above are also organized following an overall MVC software architecture for this application. This MVC-oriented organization is shown below in Table 2.

Level 1	Level 2
Model Module	ChestXRayRead
	ResultsGen
	RepCompGen
	DatabaseOps
	UserAuthMgmt
	MedInstInter
View Module	Login
	PerfScan
	ViewResults
Controller Module	AIModel
	NLPModel
	Backend
	AppController
	AppGUI

Table 2: Module MVC 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 3.

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. *Software Engineering* means the module will be implemented by the Software Engineering 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 Module

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

7.1.1 MedInstInter Module (M9)

Secrets: The data structures and algorithms used to interface with the IT system(s) of the medical institution(s).

Services: Serves as an interface to transfer information between the medical institution(s)' IT system(s) and the application.

Implemented By: [Your Program Name Here]

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 ChestXRayRead Module (M2)

Secrets: The chest x-ray reader data structure(s) and algorithm(s) used to scan a chest x-ray image and look for/detect the possible presence of certain diseases/infections (after converting from DICOM to JPEG).

Services: Reads/scans the chest x-ray image, performs some analysis and returns the processed disease/infection probability data.

Implemented By: [Your Program Name Here]

Type of Module: Library

7.2.2 ResultsGen Module (M3)

Secrets: The data structure(s) and algorithm(s) used to further process, interpret and store the results generated from scanning the chest x-ray image.

Services: Process ChestXRayRead Module output, interpret the processed disease/infection probability data and return generated results for use in generating the radiology report (components).

Implemented By: [Your Program Name Here]

Type of Module: Library, Abstract Object

7.2.3 RepCompGen Module (M5)

Secrets: The data structure(s) and algorithm(s) (including NLP) used to generate the radiology report (components) using the chest x-ray read/scan results.

Services: Take the AIModel Module output and generate the radiology report (components) documenting the results/predictions from analyzing the chest x-ray image using natural language.

Implemented By: [Your Program Name Here]

Type of Module: Library, Abstract Object

7.2.4 DatabaseOps Module (M7)

Secrets: The data structure(s) and algorithm(s) used to organize, store and retrieve patient data and application user data.

Services: Organizes, stores and retrieves patient data from the database. Also manages the chest x-ray image and scan results and predictions for each patient. Does likewise for application user data.

Implemented By: [Your Program Name Here]

Type of Module: Record, Library, Abstract Object

7.2.5 UserAuthMgmt Module (M8)

Secrets: The data structure(s) and algorithm(s) used to authenticate and manage application users.

Services: Protects patient data by only granting access to authenticated application users. Also manages users (i.e. add/remove users).

Implemented By: [Your Program Name Here]

Type of Module: Library, Abstract Object

7.2.6 Login Module (M12)

Secrets: The data structure(s) and algorithm(s) used to show and make functional the login functionality for users.

Services: Displays login portal and authenticates user logins into this application.

Implemented By: [Your Program Name Here]

Type of Module: Library, Abstract Object

7.2.7 PerfScan Module (M13)

Secrets: The data structure(s) and algorithm(s) used to show and make functional the chest x-ray read/scan functionality for users.

Services: Displays portal for users to submit chest x-ray images for read/scan to get disease/infection predictions.

Implemented By: [Your Program Name Here]

Type of Module: Library, Abstract Object

7.2.8 ViewResults Module (M14)

Secrets: The data structure(s) and algorithm(s) used to show and make functional the view results functionality for users.

Services: Displays the results of the (latest) chest x-ray read/scan for a given patient to the user (i.e. displays the latest radiology report).

Implemented By: [Your Program Name Here]

Type of Module: Library, Abstract Object

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 AIModel Module (M1)

Secrets: The data structure(s) and algorithm(s) used (including in Modules M2 ChestXRayRead and M3 ResultsGen) to read/scan chest x-ray images and interpret the disease/infection probability results. The ROC values used to interpret the disease/infection probability results and give predictions.

Services: Reads/scans chest x-ray images, interpret the data and return disease/infection predictions.

Implemented By: [Your Program Name Here]

Type of Module: Library, Abstract Object

7.3.2 NLP Module (M4)

Secrets: The data structure(s) and algorithm(s) used (including in Module M5 RepCompGen) to generate a natural language radiology report to document the results and predictions of the chest x-ray read/scan.

Services: Generates the readiology report documenting the results and predictions of the chest x-ray read/scan using natural language.

Implemented By: [Your Program Name Here]

Type of Module: Library, Abstract Object

7.3.3 Backend Module (M6)

Secrets: The data structure(s) and algorithm(s) used (including in Modules M7 DatabaseOps, M8 UserAuthMgmt and M9 MedInstInter) to manage and protect patient data, authenticate users and interface with external IT system(s) securely.

Services: Organize, store and retrieve patient data, protect patient data integrity and privacy by authenticating application users before granting access to sensitive information. Interface with IT system(s) of medical institutions to transfer/retrieve patient data securely.

Implemented By: [Your Program Name Here]

Type of Module: Record, Library, Abstract Object

7.3.4 AppController Module (M10)

Secrets: The interactions and inter-dependencies of all other software decision modules (M1 AIModel, M4 NLPModel, M6 Backend and M11 AppGUI) of this application.

Services: Enables the users to submit chest x-ray images, performs scans/reads of those images and produces disease/infection predictions that are documented in (a) natural language radiology report (components).

Implemented By: [Your Program Name Here]

Type of Module: Library

7.3.5 AppGUI Module (M11)

Secrets: The data structure(s) and algorithm(s) used (including in Modules M12 Login, M13 PerfScan and M14) to show application functionalities to the users, accept user inputs and show outputs.

Services: Show application functionalities to the users in a GUI, accept user inputs (i.e. chest x-ray images) and show outputs (i.e. predictions and radiology report/components).

Implemented By: [Your Program Name Here]

Type of Module: Library

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
FR1	M2, M13
FR2	M2, M13
FR3	M3, M14
FR4	M3, M7, M14
FR5	M7, M11
FR6	M1
FR7	M9, M10, M11
FR8	M2
FR9	M6, M7
FR10	M8, M12
FR11	M10, M11, M14

Table 3: Trace Between Requirements and Modules

AC	Modules
AC1	M1, M2, M3, M4, M5, M6, M7, M8, M9, M10, M11, M12, M13, M14
AC2	M1
AC3	M4
AC4	M11
AC5	M9

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.

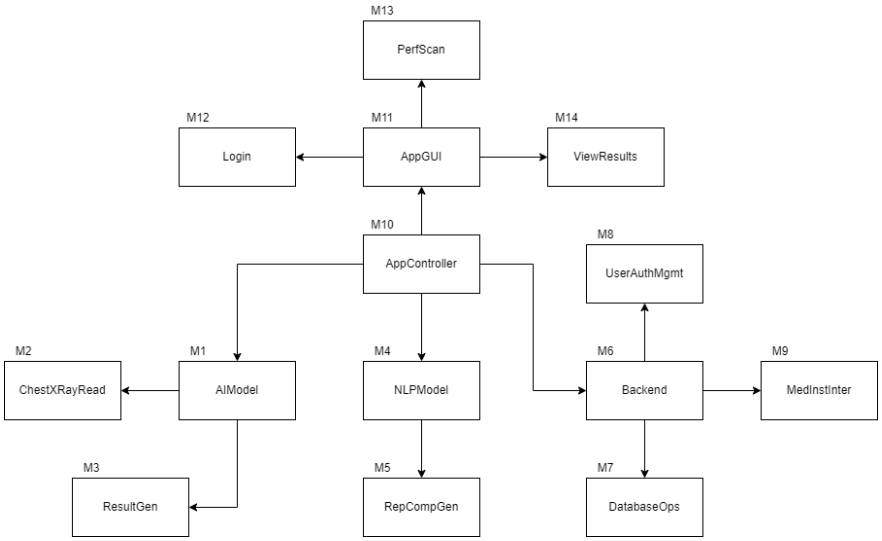

Figure 1: Use hierarchy among modules

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.