Module Guide for SFWRENG 4G06A

Team #25, RapidCare
Pranav Kalsi
Gurleen Rahi
Inreet Kaur
Moamen Ahmed

April 2, 2025

1 Revision History

Date	Version	Notes
Jan 9, 2025	1.1	Initial Document
Jan 17,2025	1.2	Revised Document Incorporating Feedback
April 1, 2025	1.2	Further revisions
April 1, 2025	1.3 Update to incorporate feedback and changes in requirements of the project.	
April 1, 2025	1.4	Updated Tracability matrix.

2 Reference Material

This section records information for easy reference.

2.1 Abbreviations and Acronyms

symbol	description
AC	Anticipated Change
DAG	Directed Acyclic Graph
M	Module
MG	Module Guide
OS	Operating System
R	Requirement
SC	Scientific Computing
SRS	Software Requirements Specification
SFWRENG 4G06A	Explanation of program name
UC	Unlikely Change
JWT	JSON Web Tokens
MFA	Multi-Factor Authentication
UI	User Interface

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 system should support different authentication methods, such as password-based, biometric login etc.

AC2: The transcription module needs to be optimized to enhance the transcription speed from audio data to written text in real-time.

AC3: The diagnosis and treatment plan prediction module should be updated with the latest medical knowledge based on the new medical research.

AC4: The system shall enable real-time streaming of audio input to written text to ensure immediate transcription without delays.

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: The requirement to secure patient's profile ensures confidentiality will remain a constant priority in the system.

UC2: The ability of the system to be compatible with the latest versions of different operating systems such as Windows, Linux and macOS will remain a constant requirement.

UC3: The requirement to exclude the background noise while using transcription module is unlikely to change for accurately classification of the medical data.

UC4: The system's ease of use is anticipated to remain a constant requirement to allow them to focus on patient's care instead of struggling with the technology.

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: User Authentication Module

M2: Administrator View Module

M3: Patient View Module

M4: Administrator Model Module

M5: Patient Model Module

M6: Service Layer Module

M7: Data Layer Module

M8: Transcription Module

M9: Classification Module

M10: Diagnosis and Treatment Plan Prediction Module

M11: AI Assistant 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 4.

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. SFWRENG 4G06A means the module will be implemented by the SFWRENG 4G06A 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.

Level 1	Level 2
Hardware-Hiding Module	None
Behaviour-Hiding Module	M1
	M2
	M3
	M4
	M5
	M6
	M7
Software Decision Module	M8
	M9
	M10
	M11

Table 1: Module Hierarchy

7.1 Hardware Hiding Modules

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 User Authentication Module (M1)

Secrets: The implementation details of the authentication mechanism, including secure storage and validation of user credentials, and session management.

Services: Authenticates users to grant access to the system based on credentials.

Implemented By: Firebase, TypeScript

Type of Module: Abstract Object

7.2.2 Administrator View Module (M2)

Secrets: Internal UI customization based on the administrative tools and functionality required by healthcare networks.

Services: Provide healthcare network adminstrators with tools to onboard, update, and remove their network on the system. Provide healthcare network adminstrators with tools to add healthcare professionals under a healthcare network to the system.

Implemented By: TypeScript, React

Type of Module: Abstract Object

7.2.3 Patient View Module (M3)

Secrets: Internal UI customization based on the tools and functionality required by healthcare professionals.

Services: Provides healthcare professionals with tools to login, create, update, and delete patient records, provide diagnostic suggestions, and medication suggestions.

Implemented By: TypeScript, React

Type of Module: Abstract Object

7.2.4 Adminstrator Model Module (M4)

Secrets: Represents the healthcare network information and characteristics as a data structure.

Services: Provides a contract of what is stored in the adminstrator account database and displayed on the UI. Any update in healthare network information on the UI can be reflected in the data model and can directly update the corresponding information in the adminstrator account database.

Implemented By: TypeScript

Type of Module: Abstract Object

7.2.5 Patient Model Module (M5)

Secrets: Represents the patient information and characteristics as a data structure.

Services: Provides a contract of what is stored in the patient account database and displayed on the UI. Any update in patient information on the UI can be reflected in the data model and can directly update the corresponding information in the patient account database.

Implemented By: TypeScript

Type of Module: Abstract Object

7.2.6 Service Layer Module (M6)

Secrets: The routing interactions between frontend and backend, as well as performing event listening and syncronization.

Services: This module provides:

- Socket communication, and event listening between the frontend and services.
- Routes requests to microservices.

Implemented By: TypeScript

Type of Module: Abstract Object

7.2.7 Data Layer Module (M7)

Secrets: Database schema for indexing strategies, and data caching mechanisms for efficient storage and retrieval of various collection.

Services: Manages secure storage, retrieval, and updates of data related to healthcare networks and healthcare professionals which are contained in collections. Request are made from the service layer, indication the collection and data to be modified.

Implemented By: Firebase

Type of Module: Record

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 Transcription Module (M8)

Secrets: The algorithm used to convert audio data into written text.

Services: Accurately converts the audio data from the conversation into written text.

Implemented By: Python

Type of Module: Library

7.3.2 Classification Module (M9)

Secrets: The algorithm used to classify the medical data after the transcription module.

Services: Accurately classifies the medical data received from the transcription module into relevant categories.

Implemented By: Python

Type of Module: Library

7.3.3 Diagonsis and Treatment Plan Prediction Module (M10)

Secrets: The retrieval and generation chain used to predict possible diagnoses and treatment plans.

Services: Predicts a applicable diagnosis along with a treatment plan based on the patient-physician conversation.

Implemented By: Python

Type of Module: Abstract Object

7.3.4 AI Assistant Module (M11)

Secrets: The algorithms used to predict possible diagnoses.

Services: Provides a assistant to pull information regarding a patient. This includes but is not limited to patient symptoms, recent diagnoses, and other demographic information.

Implemented By: Python

Type of Module: Abstract Object

8 Traceability Matrix

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

Req.	**Modules**
FR1	M2, M6, M7
FR2	M2, M6, M7
FR3	M2, M6, M7
FR4	M2, M6, M7
FR5	M2, M6, M7
FR6	M2, M6, M7
FR7	M1, M6, M7
FR8	M1, M3, M5, M6, M7
FR9	M1, M3, M5, M6, M7
FR10	M3, M5, M6, M7
FR11	M3, M5, M6, M8, M9, M7
FR12	M3, M6, M10
FR13	M3, M6, M10
FR14	M3, M11

Table 2: Trace Between Functional Requirements and Modules

Req.	**Modules**
NFR1	M2, M3
NFR2	M2, M3
NFR3	M8
NFR4	All
NFR5	All
NFR6	M1, M7
NFR7	M3, M2, M9, M10, M11
NFR8	All
NFR9	All

Table 3: Trace Between Non-Functional Requirements and Modules

Req.	Modules
AC1	M6
AC2	M1, M6, M2
IR1	M1, M6, M7
IR2	All
IR3	M10
IR4	M7
IR7	M3, M8

Table 4: Trace Between Safety and Security Requirements and Modules

AC	Modules
AC1	M1
AC2	M8
AC3	M10
AC4	M6, M3, M8, M9

Table 5: 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.

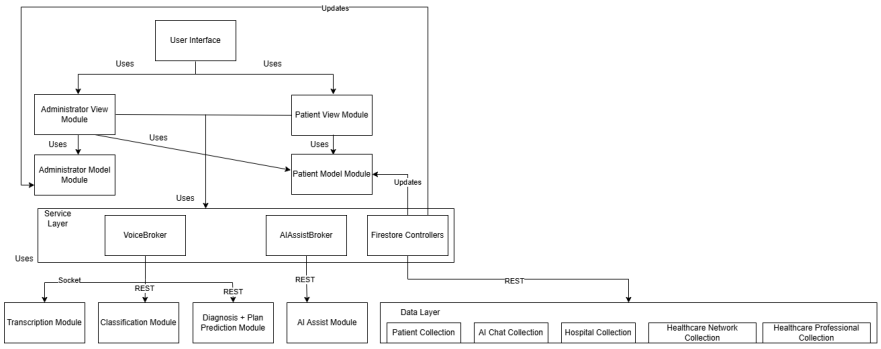

Figure 1: Use hierarchy among modules

10 User Interfaces

The user interfaces for the system are designed to provide an intuitive and efficient user experience for all users, including administrators and healthcare professionals. Below are the key interfaces:

10.1 Login Interface

Purpose: Enable users to Login to the system.

Visual Design: Include input boxes to enter login credentials and signin and signup button to login or create an account respectively.

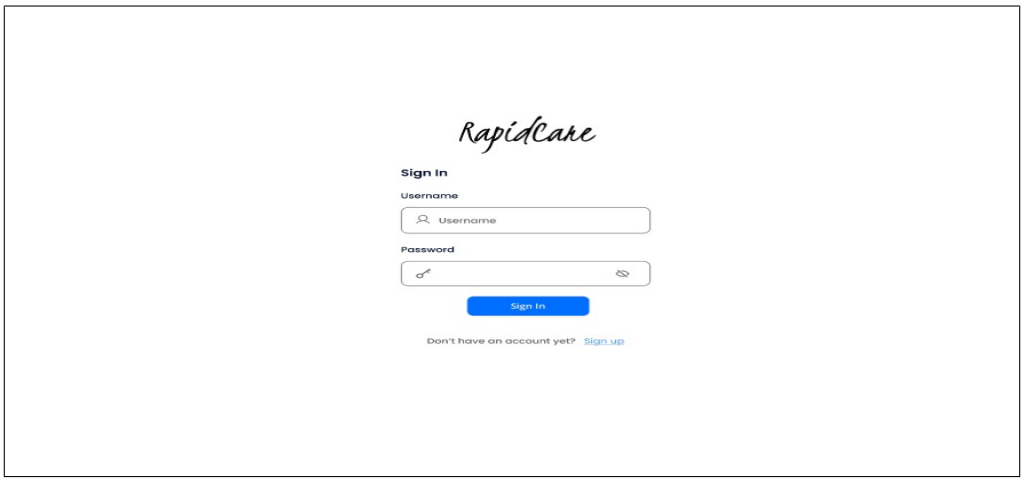

Figure 2: Login Interface

10.2 Administrator Dashboard Interface

Purpose: Enable administrators to manage healthcare network information, view metrics, and manage healthcare professionals.

Visual Design: Include a navigation bar with options to manage healthcare network, add healthcare professional, reports, and settings. Different menus with an overview of heathcare network metrics.

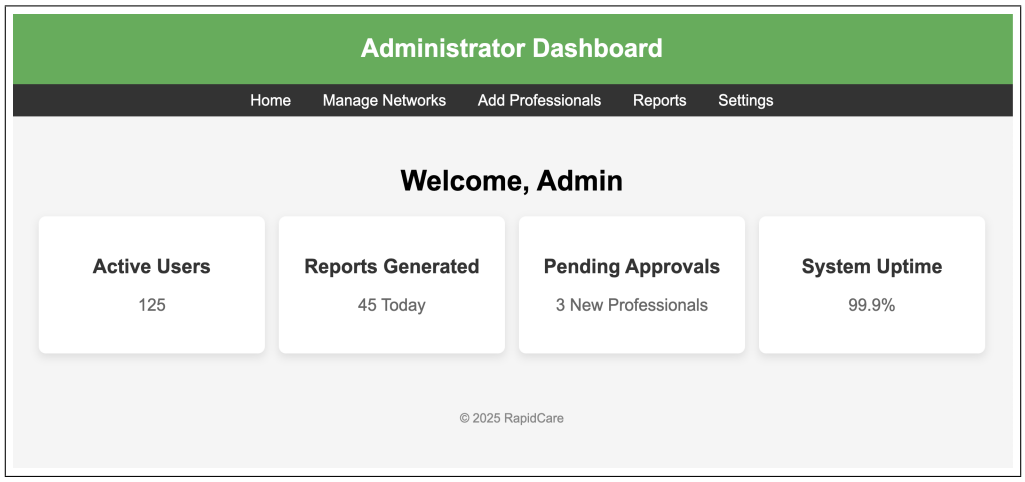

Figure 3: Administrator Dashboard Interface

10.3 Healthcare Professional Dashboard Interface

Purpose: Enable healthcare professionals to view metrics, managing patient records, and upcoming appointments.

Visual Design: Include a navigation bar with options to view appointments, patient records, notifications, and account information. Different menus with an overview of patient metrics and upcoming appointments.

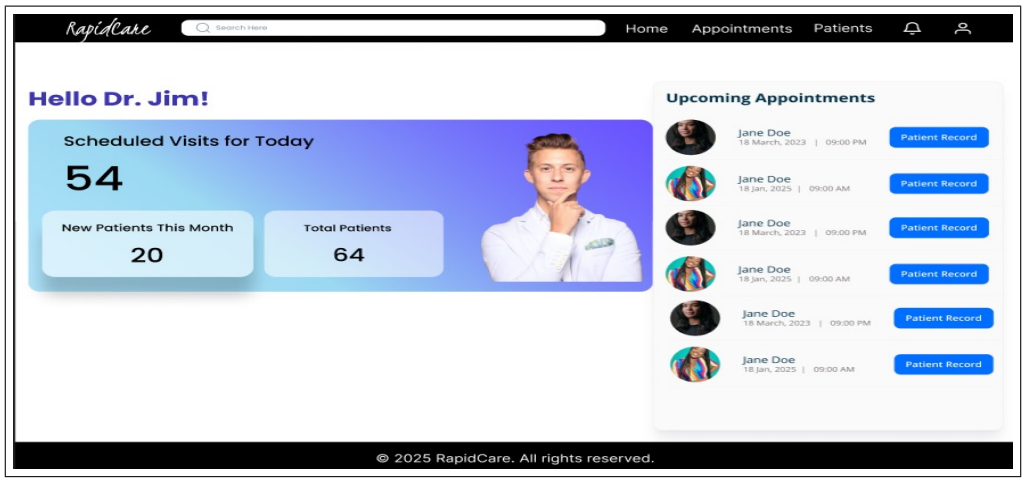

Figure 4: Healthcare Professional Dashboard Interface

10.4 Patient Profile Interface

Purpose: Enable healthcare professionals to view and edit patient record.

Visual Design: Include a breif overview of the profile and options to edit and close profile. Include a side bar to navigate quickly to frequently used options such as adding a SOAP note, view lab reports etc. Include a navigation bar with options to view profile information, medical history, consultation notes, appointment history, and documents.

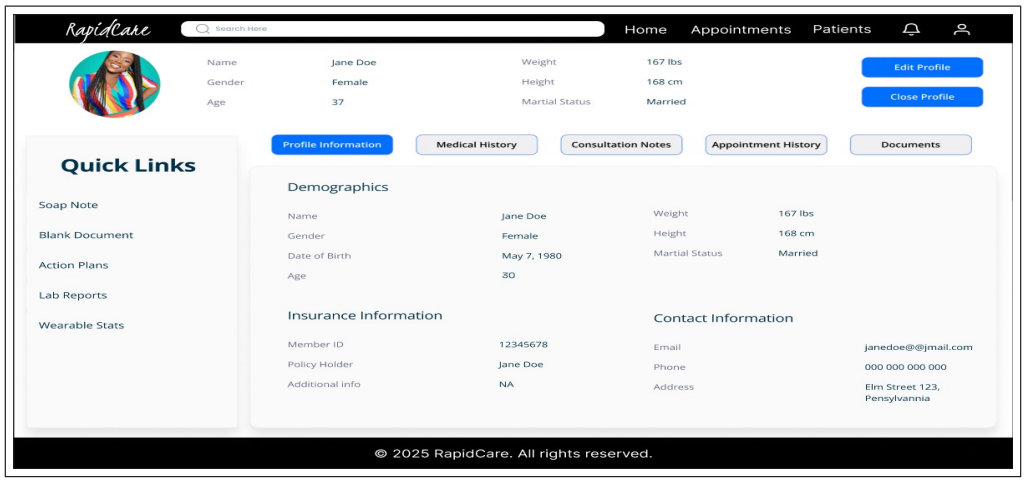

Figure 5: Patient Profile Interface

10.5 SOAP Note Interface

Purpose: Enable healthcare professionals add a SOAP note during a consultation and add it to patient profile.

Visual Design: Include different fields to add subjective assessment, objective assessment, and assessment summary. Provide different buttons to start recording, stop recording, edit note manually by typing, save note, and go back to profile.

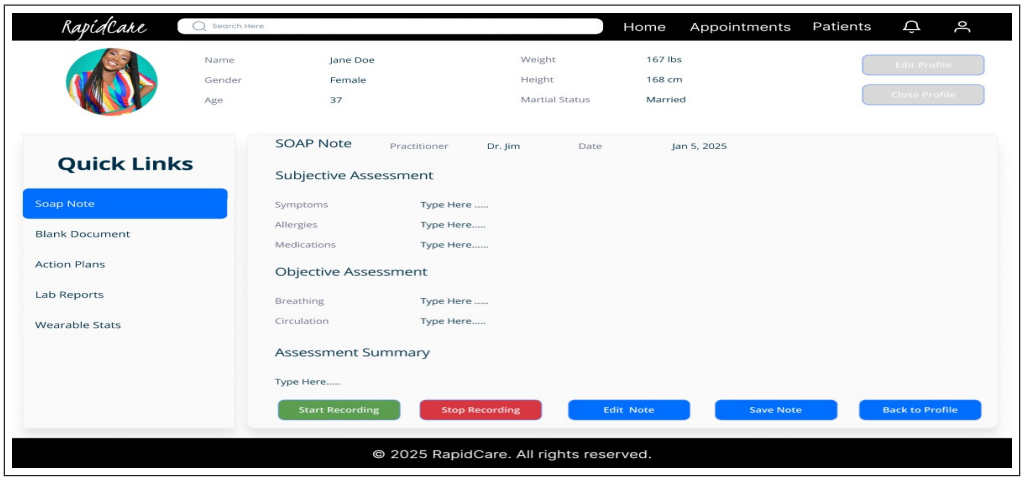

Figure 6: SOAP Note Interface

11 Design of Communication Protocols

Due to the broker architectures connecting with various services, the communication protocols differ slightly. For most of the communication as they will be simply 'POST' or 'GET'. Examples of this communication being used would be fetching patient data from the database, classifying the transcription, or employee retrieval, etc... HTTP communication will be used in **all** but one use case.

The one use case where HTTP communication will not be used is for the transcription service. For this service we will be using web socket communication as this will provide streaming capabilities for real time transcription. The frontend will act as the client who will send audio chunks to the backend to be transcribed. This will facilitate bidirectional communcation as well.

12 Timeline

- Week of Jan. 19th – Rev0 Demonstration:

  - Transcription optimization (Gurleen, Pranav)
  - Text Classification Service Review (Moamen)
  - Setup Socket communication through API and Transcription Service (Pranav)
  - Containerize application + Setup deployment environment (Pranav)
  - Build out login page, dedication classify text UI (Inreet)
- Week of Jan. 26th
  - Setup ML Training Pipeline (Pranav)
  - Employee management + Patient Management Views (Inreet, Gurleen)
  - Data Layer setup (Moamen)
- Week of Feb. 2nd
  - Rev0 Demonstration
  - Complete final optimizations and refinements (All)
  - Worker onboarding FR
- Week of Feb. 9th
  - Supervisor acceptance meeting + resolve supervisor reviews (All)
- Week of Feb. 16th
  - VnV Report Completion
  - Testing introduce testing suites as per VnV.
- Week of Feb. 23rd
  - VnV Report Completion
- Week of Mar. 2nd
  - VnV Report Submission
- Week of Mar. 9th
  - Complete outlying tickets converse with supervisor on additional functionality.
  - Suggestion model retraining capabilities.
- Week of Mar. 16th
  - Diagonistic suggestions and Medication suggestions
  - Usable API for other networks to send patient charts.

  - Presentation preparation

- Week of Mar. 23rd
  - Presentation preparation finishing

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.