Module Guide for Natural Language Processing for Mental Health Risk Prediction

Team 13, The Cognitive Care Crew
Jessica Dawson
Michael Breau
Matthew Curtis
Benjamin Chinnery
Yaruo Tian

February 16, 2024

1 Revision History

Date	Version	Notes
January 17, 2024	1.0	Revision 0

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
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 5 lists the anticipated and unlikely changes of the software requirements. Section 6 summarizes the module decomposition that was constructed according to the likely changes. Section 7 specifies the connections between the software requirements and the modules. Section 8 gives a detailed description of the modules. Section 9 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 10 describes the use relation between modules.

4 General Structure of an NLP Pipeline

All NLP pipelines in this system can be thought of as following a general structure. This is not a hard description of the structure each diagnosis component will use but a soft description of what the system's NLP pipelines will look like. This pattern is likely to show up throughout the system in various locations and is included here to provide context and a conceptual model the reader can use for the rest of the document. The proposed structure is as follows:

Input $\rightarrow$ Tokenizer $\rightarrow$ Representation Model $\rightarrow$ Prediction Model $\rightarrow$ Predictions

Where the Prediciton Model can be further broken up into:

$\rightarrow$ Transformation $\rightarrow$ ... $\rightarrow$ Transformation $\rightarrow$

A brief description of each stage is as follows:

- Input: The input text to the pipeline.
- Tokenizer: Responsible for "cleaning" the text, can perform many different tasks: removing junk words (often referred to as stopwords) like the, and, a, etc.; expanding contractions; removing numerics and URLS. Leaves the text as text, just with the less useful features removed.
- Representation Model: Machine learning models can not directly operate on text, they need numerical inputs. The representation model converts from the text to some numerical format. Different representation models output different formats and encode different information into these formats.
- Prediction Model: Transforms the numerical format provided by the representation model into the mental health predictions of interest. The prediction model may transform the data mutliple times in different ways before reaching a final prediction.
  - Transformation: A specific transformation in the prediction model's pipeline
- Predictions: The final predictions produced by the pipeline.

Due to the interconnected nature of this pipeline and the research based nature of this project large changes to this pipeline are both expected and unavoidable. This is because changes in one section of pipeline can require large changes in other sections, trying a new prediction model may require an entirely different representation model do what it's meant tobe used.

5 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 5.1, and unlikely changes are listed in Section 5.2.

5.1 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: What format the input data takes.

AC2: What format the output data takes.

AC3: What stages there are in a task's NLP pipeline.

AC4: How the raw text data is "cleaned" (see 4: tokenizer).

AC5: How the cleaned text data is converted into numerical data.

AC6: How many manipulations of the numerical data are performed before producing a prediction.

AC7: How the numerical data is manipulated to produce predictions.

AC8: How the system will integrate with the tools the operations team is developing.

5.2 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, Output: File).

UC2: What diagnosis tools the system will provide.

6 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. As the details of the NLP pipelines are not as of yet determined some modules are left quite general and vague (ie. how many transformation modules will be used, what representation models will be used, etc.).

M1: Hardware-Hiding Module

M2: Search for Symptoms of Depression Module

M3: Early Risk Detection for Symptoms of Anorexia Module

EDM1: ED Input

EDM2: ED Output

EDM3: ED Pipeline Manager

EDM4: ED Tokenizer

EDM5: ED Representation Model

EDM6: ED Prediction Model

EDM7: ED Transformation 1...n

Level 1	Level 2	Level 3
Hardware-Hiding	None	
Behaviour-Hiding		
	Eating Disorder IO	ED Input ED Output
Software Decision	Depression Module Anorexia Module	
	Eating Disorder Pipeline	ED Pipeline Manager ED Tokenizer ED Representation Model ED Prediction Model ED Transformation 1...n (represents multiple modules)

Table 1: Module Hierarchy

7 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.

8 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. *Natural Language Processing for Mental Health Risk Prediction* means the module will be implemented by the Natural Language Processing for Mental Health Risk Prediction 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.

8.1 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

8.2 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: Depression IO, Anorexia IO, Eating Disorder IO

8.2.1 Eating Disorder IO (EDM1,EDM2)

Secrets: How input and output is handled for the eating disorder diagnosis tool.

Services: Includes programs that provide reading from an input file and outputting to an output file for the eating disorder diagnosis tool.

Implemented By: ed_input.py, ed_output.py

8.2.1.1 ED Input (EDM1)

Secrets: What format the input file is provided in.

Services: Provides functionality for reading user posts from a file and formatting them into a data structure usable by the rest of the system.

Implemented By: ed_input.py

Type of Module: Library

8.2.1.2 ED Output (EDM2)

Secrets: What format the output file is written in.

Services: Provides functionality for writing predictions to a file.

Implemented By: ed_output.py

Type of Module: Library

8.3 Software Decision Module

8.3.1 Search for Symptoms of Depression Module (M2)

Secrets: The format and structures of detecting symptoms of depression data objects.

Services: Sorts and ranks 1000 sentences of users to a BDI Questionnaire with 21 questions in correspondence. Each question of the questionnaire will correspond to 1000 related sentences.

Implemented By: Natural Language Processing for Mental Health Risk Prevention

8.3.2 Early Risk Detection for Symptoms of Anorexia Module (M3)

Secrets: The Natural Language processor which will analyze inputted user data to determine a risk for anorexia

Services: Anorexia training data do not provide direct interaction with the user.

Implemented By: Natural Language Processing for Mental Health Risk Prevention

8.3.3 Eating Disorder Pipeline (-)

Secrets: The structure and design of the eating disorder NLP pipeline.

Services: Includes programs that manage the overall pipeline and others that define the specific transformations used to produce a prediction.

Implemented By: ED Pipeline Manager, ED Tokenizer, ED Representation Model, ED Prediction Model, ED Transformation 1...n

8.3.3.1 ED Pipeline Manager (EDM3)

Secrets: What components the NLP pipeline for the eating disorder task is comprised of.

Services: Orchestrates and runs all the components of the eating disorder NLP pipeline.

Implemented By: ed_pipeline.py

Type of Module: Library

8.3.3.2 ED Tokenizer (EDM4)

Secrets: How the raw text data is cleaned in preparation of the model.

Services: Provides a routine for the cleaning and stripping of less useful information from raw text.

Implemented By: ed_tokenizer.py

Type of Module: Library

8.3.3.3 ED Representation Model (EDM5)

Secrets: How text data is converted into a numerical format.

Services: Provides a routine for converting text data into a numerical format.

Implemented By: ed_rep_model.py

Type of Module: Library

8.3.3.4 ED Prediction Model (EDM6)

Secrets: The overall structure of the transformations used to produce a prediction from the numerical representation of the text.

Services: Orchestrates and runs all the transformations that make up the prediction model.

Implemented By: ed_pred_model.py

Type of Module: Library

8.3.3.5 ED Transformation 1...n (EDM7)

Note: Currently represents multiple modules, when a clearer picture of what structure the eating disorder pipeline will take on is reached this will be updated and expanded into multiple specific transformations

Secrets: What transformations are used to produce a prediction from the numerical representation of the text.

Services: Provides a routine for transforming numerical data from one form to another.

Implemented By: ed_transformation1...n.py

Type of Module: Library

9 Traceability Matrix

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

Req.	Modules
T1FR-1	M2
T1FR-2	M2
T1FR-3	M2
T2FR-1	M3
T2FR-2	M3
T2FR-3	M3
T2FR-4	M3
T3FR-1	EDM1
T3FR-2	EDM2
T3FR-3	EDM2
GR1	M2, M3, EDM1, EDM2
GR2	M2, M3, EDM2
SR1	M2, M3, EDM2
SR2	M2, M3, EDM2

Table 2: Trace Between Requirements and Modules

AC	Modules
AC1	M2, M3, EDM1
AC2	M2, M3, EDM2
AC3	M2, M3, EDM3
AC4	M2, M3, EDM4
AC5	M2, M3, EDM5
AC6	M2, M3, EDM6
AC7	M2, M3, EDM7
AC8	M2, M3

Table 3: Trace Between Anticipated Changes and Modules

10 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

11 Timeline

- Module: EDM1 Finish Date: Jan 20 2024 Assignee(s): Jessica
- Module: EDM2 Finish Date: Jan 22 2024 Assignee(s): Jessica
- Module: EDM3 Finish Date: Jan 24 2024 Assignee(s): Jessica
- Module: EDM4 Finish Date: Jan 26 2024 Assignee(s): Jessica
- Module: EDM5 Finish Date: Jan 28 2024 Assignee(s): Jessica
- Module: EDM6 Finish Date: Jan 28 2024 Assignee(s): Jessica
- Module: EDM7 Finish Date: Feb 1 2024 Assignee(s): Jessica
- Module: M2 Finish Date: Feb 1 2024 Assignee(s): Franklin, Matthew
- Module: M3 Finish Date: Feb 1 2024 Assignee(s): Michael, Ben
- Module: M1 Finish Date: Feb 1 2024 Assignee(s): All
- Testing: All Modules Finish Date: Feb 4 2024 Assignee(s): All

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.