yoGERT GIS Toolbox

Capstone 4G06

Module Guide for yoGERT

Team 19,
Smita Singh, Abeer Alyasiri, Niyatha Rangarajan,
Moksha Srinivasan, Nicholas Lobo, Longwei Ye

April 6, 2023

1 Revision History

Date	Version	Notes
1/17/2023	1.0	**Longwei**, Raw Documentation Generated & Contents filled in
1/18/2023	1.1	**Longwei**, Modules update based on MIS document
1/18/2023	1.2	**Abeer**, Correct Module Decomposition
1/18/2023	1.3	**Abeer and Longwei**, Review of the MG document
1/18/2023	1.3	**Longwei**, Add NFRs to trace table
4/4/2023	1.4	**Longwei**, Update MG for REV1

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
O/S	Out of scope
R	Requirement
SC	Scientific Computing
SRS	Software Requirements Specification
	Explanation of program name
UC	Unlikely Change
yoGERT	GPS data points analyze library implemented on Python

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 mechanism or functionality of the program.

AC3: The format of the output data

AC4: The online databases and libraries that modules use for implementing secrets.

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 format of the input data.

UC3: There always exists open source data for the program's modules.

UC4: The graph of nodes that shows the connections between GPS points.

UC5: The way that the user interacts with the program. For example, users call the methods provided in the yoGERT library to do data analyze.

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: Hardware-Hiding Module

M2: Preprocessing Inputs Module

M3: Network Graph Module

M4: Generate Episodes Module

M5: ~~Shortest Route Module~~

M6: Shortest Route Episode Module

M7: Shortest Route Stop Module

M8: Shortest Route Trace Module

M9: Alternative Route Module

M10: ~~Route Generation Module~~

M11: Data Transformation Module

M12: Fetch Activity Locations Module

M13: Mapping Module

M14: ~~Main 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 2.

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. means the module will be implemented by the 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	
Behaviour-Hiding Module	M4 ~~M10~~ M12 M13 ~~M14~~
Software Decision Module	M2 M3 ~~M5~~ M6 M7 M8 M9 M11

Table 1: Module Hierarchy

7.1 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 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. Also, it provides functionality for the user to interact with the system.

Implemented By: OS ~~(Linux)~~

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: N/A

7.2.1 Generate Episode Module (M4)

Secrets: Episode generation algorithm

Services: Filtering the input data points and keeps the points that are significant to the user's session. In addition, episode activity mode (i.e. stop, walk, drive) are added for each segment that survives from filtering.

Implemented By: ~~Python~~yoGERT

Type of Module: Abstract Object

7.2.2 ~~Route Generation Module (M10)~~

Secrets: ~~Inputs~~

Services: ~~Transfer the filtered dataset into specific dataform for route analysis.~~

Implemented By: ~~PythonyoGERT~~

Type of Module: ~~Library.~~

7.2.3 Activity Locations Generation Module (M12)

Secrets: Activity location fetching algorithm, ~~output~~

Services: Allows the user to navigate through a list of possible locations within the tolerance radius of the given input stop GPS points generated by the program

Implemented By: ~~Python~~yoGERT

Type of Module: Library

7.2.4 Mapping Module (M13)

Secrets: ~~Output,~~ Map generation

Services: Deliver an interactive map that consists of information(i.e. , activity location, and points) generated by the previous modules.

Implemented By: ~~Python~~yoGERT, html

Type of Module: Abstract Object

7.2.5 ~~Main Module (M14)~~

~~**Secrets:** User Interaction~~

~~**Services:** Allow the user to call all yoGERT's functions(i.e. generate map, generate route, preprocess data, activity points, or episodes)~~

~~**Implemented By:** Python~~

~~**Type of Module:** Library~~

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: N/A

7.3.1 Preprocessing Inputs Module (M2)

Secrets: Inputs

Services: Clean and validate the CSV file input.

Implemented By: ~~Python~~yoGERT

Type of Module: Library.

7.3.2 Network Graph Module (M3)

Secrets: Inputs

Services: Allows user to generate a NetworkGraph object that consists of network consisting of nodes and edges based on specified travel mode(i.e. walk, drive) for the motion.

Implemented By: ~~Python~~yoGERT

Type of Module: Abstract Object.

7.3.3 ~~Shortest Route Module (M5)~~

~~**Secrets:** Route finding algorithm~~

~~**Services:** Generate the shortest route from one point to another compatible with NetworkGraph object.~~

~~**Implemented By:** Python~~

~~**Type of Module:** Abstract Object.~~

7.3.4 Shortest Route Episode Module (M6)

Secrets: Route finding algorithm

Services: Generate the shortest route from one Episode consists of travel points compatible with NetworkGraph object.

Implemented By: yoGERT

Type of Module: Abstract Object.

7.3.5 Shortest Route Stop Module (M7)

Secrets: Route finding algorithm

Services: Generate the shortest route from a collection of stop points compatible with NetworkGraph object.

Implemented By: yoGERT

Type of Module: Abstract Object.

7.3.6 Shortest Route Trace Module (M8)

Secrets: Route finding algorithm

Services: Generate the shortest route from one group of trace points compatible with NetworkGraph object.

Implemented By: yoGERT

Type of Module: Abstract Object.

7.3.7 Alternative Route Module (M9)

Secrets: Route generation algorithm

Services: Generate the bus route from one point to another compatible with NetworkGraph object.

Implemented By: ~~Python~~yoGERT

Type of Module: Abstract Object.

7.3.8 Data Transformation Module (M11)

Secrets: Inputs

Services: Transfer the filtered dataset into specific data forms for route analysis, activity fetching, and mapping.

Implemented By: ~~Python~~yoGERT

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
R1	~~M14~~ M2, M4, M6, M7,M8
R2	M2, M3, M4
R3	M4, M12, M6, M7,M8, M13
R4	M3, M4, M11, ~~M5~~, M6, M7,M8 M9, ~~M10~~, M12
R5	M4
R6	M4
R7	M4
R8	M4
R9	M4, M12
R10	O/S
R11	O/S, M6, M7,M8
R12	~~M10~~, partial O/S
R13	~~M10, M9~~ M12
R14	~~M4~~ M12
R15	M4, partial O/S
R16	M13
R17	M13
R18	M13
NFR1	-
NFR2	M13
NFR3	-
NFR4	~~M14~~ -
~~NFR5-NFR7~~	-
NFR8	M4, M12, M6, M7,M8
NFR9	~~M14,~~ partial O/S
~~NFR10~~	-
NFR11	-
NFR12	~~M14~~ M2, M4, M12, M6, M7,M8
~~NFR13~~	-
NFR14	M1
NFR15	M1
NFR16-NFR20	-
NFR21	M2
NFR22	M2
NFR23-NFR27	-

Table 2: Trace Between Requirements and Modules

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

Table 3: 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.

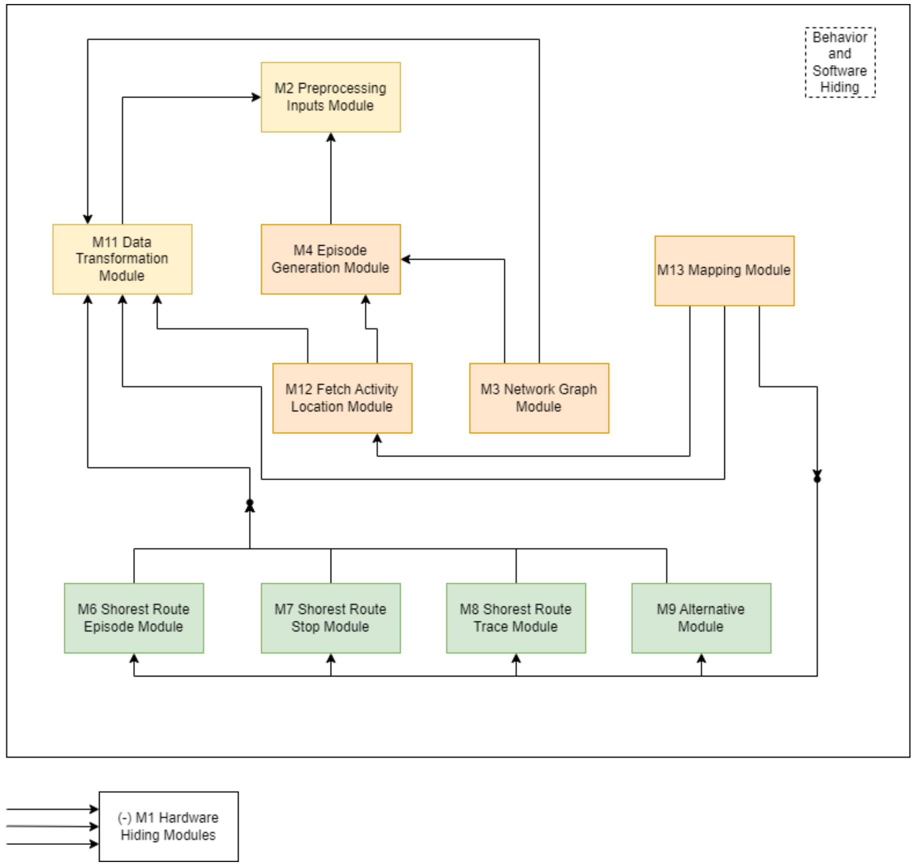

Figure 1: Use hierarchy among modules

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.