Module Guide for Centrality in Graphs

Atiyeh Sayadi

March 10, 2024

1 Revision History

Date	Version	Notes
Date 1	1.0	Notes
Date 2	1.1	Notes

2 Reference Material

This section records information for easy reference.

2.1 Abbreviations and Acronyms

symbol	description
AC	Anticipated Change
CC	Closeness Centrality
CIG	Centrality in Graphs
DAG	Directed Acyclic Graph
DC	Degree Centrality
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 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

This software, like other softwares, may undergo changes. These changes are as follows:

AC1: Changing the Initial Graph from Undirected to Directed.

AC2: Computing Other Centrality Measures such as Betweenness Centrality.

AC3: Reading a file from various inputs such as Excel.

4.2 Unlikely Changes

The future changes of the software do not include the following items:

UC1: The input matrix will still be of size n*2.

UC2: The graph representation in the output will remain unchanged. A visual depiction of the graph will still be displayed.

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: Behaviour-Hiding Module

M3: Software Decision 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.

Functional requirments:

Level 1	Level 2
Hardware-Hiding Module	-
Behaviour-Hiding Module	GUI File Graph
Software Decision Module	Matrix Output

Table 1: Module Hierarchy

FR1: Degree centrality for all nodes ranges between zero and one.

FR2: Degree centrality has been calculated for all nodes.

FR3: Closeness centrality for all nodes ranges between zero and one.

FR4: Closeness centrality has been calculated for all nodes.

Nonfunctional requirments:

NFR1: Accuracy

NFR2: Usability

NFR3: Maintainability

7 Module Decomposition

Modules are decomposed according to the principle of "information hiding" proposed by Parnas et al. (1984). The following is an introduction to the modules and a brief explanation of each of them.

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

7.2.1 File

Secrets: Storage location and file reading method.

Services: This module reads the initial graph matrix from a text file and build the initial matrix.

Implemented By: CIG

Type of Module: Abstract data type

7.2.2 GUI

Secrets: Setting up user interface settings for displaying outputs.

Services: This module utilizes the results of calculations from the Graph modules and displays the values separately on the nodes of the graph.

Implemented By: CIG

Type of Module: Abstract data type

7.2.3 Graph

Secrets: Data structure for a graph using the concept of adjacency matrix.

Services: This module provides a data structure for a graph, which consists of a set of nodes and edges. Then, based on this information, it calculates the node degree, the shortest path from one node to other nodes, degree centrality, and closeness centrality.

Implemented By: CIG

Type of Module: Abstract data type

7.3 Software Decision Module

7.3.1 Matrix

Secrets: A module for performing mathematical calculations on multi-dimensional data.

Services: Using the NumPy library, operations on matrices, such as creating matrices, loading them, calculating their size, etc., can be performed as follows.

Implemented By: Numpy

Type of Module: Library

7.3.2 Output

Secrets: Providing a graphical user interface.

Services: Using the tkinter library and its functions in Python to create a graphical user interface (GUI).

Implemented By: Tkinter

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	Graph
FR2	Graph
FR3	Graph
FR4	Graph
NFR1	Graph, File, GUI, Matrix, Output
NFR2	Output, GUI
NFR3	Graph, File, GUI, Matrix, Output

Table 2: Trace Between Requirements and Modules

AC	**Modules**
AC1	Graph
AC2	Graph
AC3	File

Table 3: Trace Between Anticipated Changes and Modules

9 Use Hierarchy Between Modules

Figure 1 illustrates a hierarchical view of the relationship between the modules.

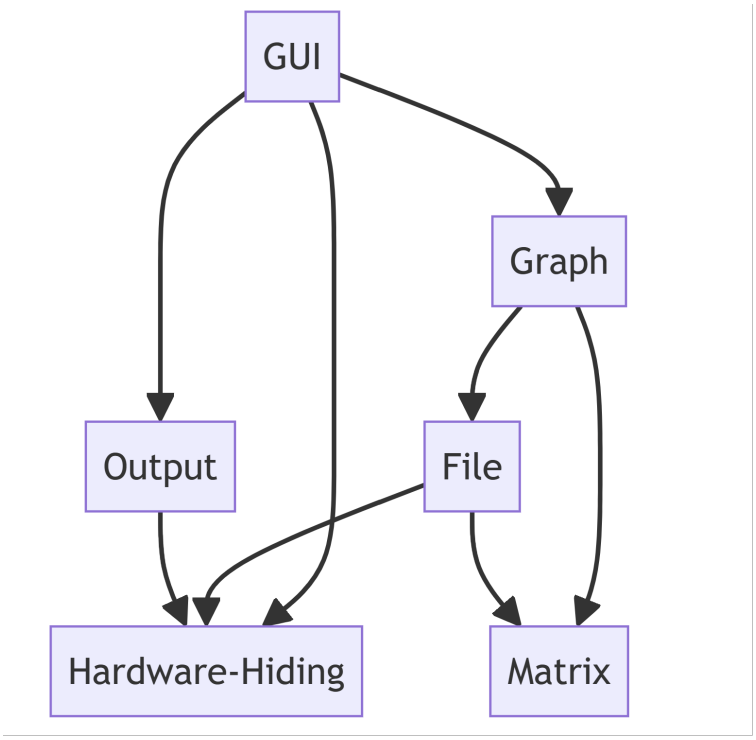

Figure 1: Use hierarchy among modules

10 User Interfaces

This software provides a simple graphical interface that displays the graph image for both centrality measures, degree centrality and closeness centrality, while indicating the respective values for each node.

11 Design of Communication Protocols

12 Timeline

Table 4 displays the timeline for the development of each module and its respective developer.

Modules	**Time**	**Responsible**
File	20-25 March	Atiyeh Sayadi
Graph	26- 31 March	Atiyeh Sayadi
GUI	1-10 April	Atiyeh Sayadi
Matrix	20- 31 March	Atiyeh Sayadi
Output	1-10 April	Atiyeh Sayadi

Table 4: Timeline

References

David L. Parnas. On the criteria to be used in decomposing systems into modules. *Comm. ACM*, 15(2):1053–1058, December 1972.

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.