Module Guide for Truss Tool

Maryam Valian

March 30, 2023

1 Revision History

Date	Version	Notes
11/03/2023	1.0	Draft version
12/03/2023	1.1	Update module decomposition
15/03/2023	1.2	Update module hierarchy
16/03/2023	1.3	Update after presentation feedback
23/03/2023	1.4	Update based on First reviewer feedback
30/03/2023	1.5	Update based on Second reviewer feedback

2 Reference Material

This section records information for easy reference.

2.1 Abbreviations and Acronyms

symbol	description
AC	Anticipated Change
DAG	Directed Acyclic Graph
IF	Internal forces of a truss
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

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 format of the initial input data.

AC3: The format of the initial input parameters.

AC4: The constraints on the input data.

AC5: The format of the final output data.

AC6: The constraints on the output data.

AC7: How to decompose forces according to input data of truss type.

AC8: How the equilibrium equations are defined from Input data.

AC9: The implementation of the sequence data structure.

AC10: The algorithm used for the linear equation solver.

Both AC9 and AC10 are related to libraries in Phyton. So in future, it may change. AC7 and AC8 are anticipated changes if the truss is considered to be three-dimensional.

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: An external source of input data should be provided for the software.

UC3: The output data are always displayed at the output devices.

UC4: The goal of the system is to calculate support reactions and internal forces for each member.

UC5: The equilibrium equations can be defined using parameters defined in the input parameter module.

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: Input parameters module

M3: Input verification module

M4: Specification parameters module

M5: Output format module

M6: Output verification module

M7: Support reaction equations module.

M8: Force decomposing module

M9: IF equations module.

M10: Control module

M11: Sequence data structure module

M12: linear equation solver module

Not all of the above list will be implemented. Since M1 is already implemented in operating system. Also M11 and M12 are already implemented in Numpy and Scipy libraries of Python, respectively.

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.

Level 1	Level 2
Hardware-Hiding Module	
Behaviour-Hiding Module	Input parameters module
	Input verification module
	Specification parameters module
	Output format module
	Output verification module
	Equilibrium equations for free body of truss module.
	Force decomposing module
	Equilibrium equations for all joints module.
	Control module
Software Decision Module	Sequence data structure module
	linear equation solver module

Table 1: Module Hierarchy

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. *Truss Tool* means the module will be implemented by the Truss Tool 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 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

7.2 Behaviour-Hiding Modules

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: Truss Tool

7.2.1 Input Format Module (M2)

Secrets: The format and structure of the input data.

Services: Converts the input data into the data structure used by the input parameters module.

Implemented By: [Truss Tool]

Type of Module: [Abstract Data Type] [Information to include for leaf modules in the decomposition by secrets tree.]

7.2.2 Input verification module (M3)

Secrets: The algorithm of the verification is isolated from the program.

Services: Checks whether the input data complies with software constraints or not. If the check fails, then an exception is thrown with an explanation of the cause.

Implemented By: [Truss Tool]

Type of Module: [Abstract Data Type] [Information to include for leaf modules in the decomposition by secrets tree.]

7.2.3 Specification parameters module (M4)

Secrets: The values of the specification parameters and constants according to SRS.

Services: Read access for the specification parameters.

Implemented By: [Truss Tool]

Type of Module: [Record] [Information to include for leaf modules in the decomposition by secrets tree.]

7.2.4 Format output module (M5)

Secrets: The format and structure of output.

Services: Outputs results of support reaction and internal forces.

Implemented By: [Truss Tool]

Type of Module: [Abstract Data Type] [Information to include for leaf modules in the decomposition by secrets tree.]

7.2.5 Output verification module (M6)

Secrets: The algorithm used to check correctness of results.

Services: Verifies that the internal forces for the last joint is correct.

Implemented By: [Truss Tool]

Type of Module: [Abstract Data Type] [Information to include for leaf modules in the decomposition by secrets tree.]

7.2.6 Support reaction module (M7)

Secrets: The algorithm used for the calculation of support reactions.

Services: Creates equilibrium equations and solves them by linear solver module.

Implemented By: [Truss Tool]

Type of Module: [Abstract Data Type] [Information to include for leaf modules in the decomposition by secrets tree.]

7.2.7 Force decomposition module (M8)

Secrets: The algorithm used for decomposition of forces.

Services: Decomposes internal forces in direction of X-axis and Y-axis.

Implemented By: [Truss Tool]

Type of Module: [Abstract Data Type] [Information to include for leaf modules in the decomposition by secrets tree.]

I'm not sure of the role of 7.2.7 & 7.2.8. I looked at the MIS and I'm still confused. Maybe you only need one module to build the system of equations for solving?

7.2.8 Internal Forces module (M9)

Secrets: The algorithm used to create equilibrium equations for each joint.

Services: Calculates all equilibrium equations needed to calculate internal forces of the truss.

Implemented By: [Truss Tool]

Type of Module: [Abstract Data Type] [Information to include for leaf modules in the decomposition by secrets tree.]

7.2.9 Control module (M10)

Secrets: The algorithm for coordinating the running of the program.

Services: Provides the main program.

Implemented By: [Truss Tool]

Type of Module: [Abstract Data Type] [Information to include for leaf modules in the decomposition by secrets tree.]

7.3 Software Decision Modules

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 Sequence data structure module (M11)

Secrets: The data structure for a sequence data type.

Services: Provides array creation and modifications.

Implemented By: [Numpy- Python] https://numpy.org/doc/stable/reference/

Type of Module: [Library]

7.3.2 Linear equation solver (M12)

Secrets: The algorithm used for solving a linear equation.

Services: Solves a linear equation.

Implemented By: [Scipy- Python] https://docs.scipy.org/doc/scipy/reference/

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	M1,M4 M2, M3, M10
R2	M1,M5,M10
R3	M5,M6,M7,M8,M9,M10,M11,M12
R4	M6,M10

Table 2: Trace Between Requirements and Modules

AC	Modules
AC1	M1
AC2	M2
AC3	M3
AC4	M6
AC5	M5
AC6	M6
AC7	M8
AC8	M7,M9
AC9	M11
AC10	M12

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. Also from left to right, there is an order of using modules by Control module.

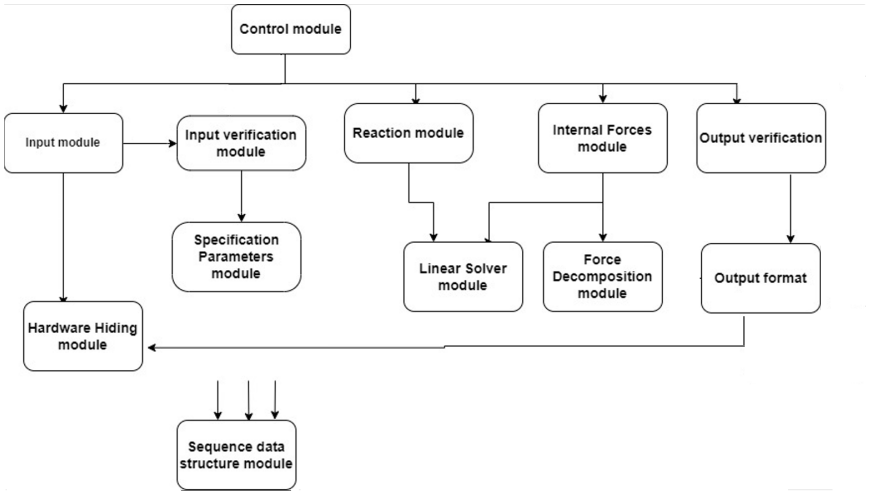

Figure 1: Use hierarchy among modules

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.