Module Guide: Breaking Effect

Marshall Xiaoye Ma

November 7, 2017

1 Revision History

Date	Version	Notes
2017-10-29	1.0	New document

Contents

List of Tables

List of Figures

2 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 used 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 3 lists the anticipated and unlikely changes of the software requirements. Section 4 summarizes the module decomposition that was constructed according to the likely changes. Section 5 specifies the connections between the software requirements and the modules. Section 6 gives a detailed description of the modules. Section 7 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 8 describes the use relation between modules.

3 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 3.1, and unlikely changes are listed in Section 3.2.

3.1 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 cutting function provided by Unity3D.

3.2 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: Keyboard, Output: Screen).

UC2: There will always be a source of input data external to the software.

UC3: Input more than one coefficient of friction on the ground.

4 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: Input Module

M4: Obtaining gravity center module

M5: Angle calculation module

M6: Displacement in the air calculation module

M7: Displacement on the ground calculation module

M8: Software Decision Module

M9: Object cutting module

M10: Output Module

Level 1	**Level 2**
Hardware-Hiding Module	
Behaviour-Hiding Module	Input verification Module Input Format Module Obtaining gravity center module Angle calculation module Displacement calculation module one Displacement calculation module two
Software Decision Module	Object cutting module Output Module

Table 1: Module Hierarchy

5 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.

6 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. Also indicate if the module will be implemented specifically for 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. Whether or not this module is implemented depends on the programming language selected.

6.1 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

6.2 Behaviour-Hiding Module(M2)

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: Breaking Effect

6.2.1 Input Module (M3)

Secrets: The format, structure verification of the input data.

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

Implemented By: Breaking Effect

6.2.2 Obtaining gravity center module(M4)

Secrets: How target object is cutting and how to obtain location of gravity centers; Structure of locations of each pieces.

Services: Call cutting function provided by unity to split target object then do a traversal to obtain location of gravity centers for all pieces.

Implemented By: Breaking Effect

6.2.3 Angle calculation module(M5)

Secrets: How the angles are calculated.

Services: Calculate the angle between initial speed v_0 and horizontal θ_1. Calculate the angle between x axiom and projection on horizontal of initial speed θ_2.

Implemented By: Breaking Effect

6.2.4 Displacement in the air calculation module(M6)

Secrets: How the displacement are calculated.

Services: Calculate and output trace of motion for each piece in the air.

Implemented By: Breaking Effect

6.2.5 Displacement calculation on the ground module(M7)

Secrets: How the displacement are calculated.

Services: Calculate and output trace of motion for each piece on the ground.

Implemented By: Breaking Effect

6.3 Software Decision Module(M8)

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: Breaking Effect

6.3.1 Object cutting Module(M9)

Secrets: How the target object is splitted into pieces.

Services: Split target object into pieces..

Implemented By: Unity3D

6.3.2 Output Module(M10)

Secrets: Interact with platform to convert data into visualization.

Services: Display motion of pieces in vision.

Implemented By: Unity3D

7 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,M3
R2	M3
R3	M3
R4	M4,M9
R5	M5
R6	M6
R7	M7

Table 2: Trace Between Requirements and Modules

AC	**Modules**
AC1	M1
AC2	M3
AC3	M9

Table 3: Trace Between Anticipated Changes and Modules

8 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.

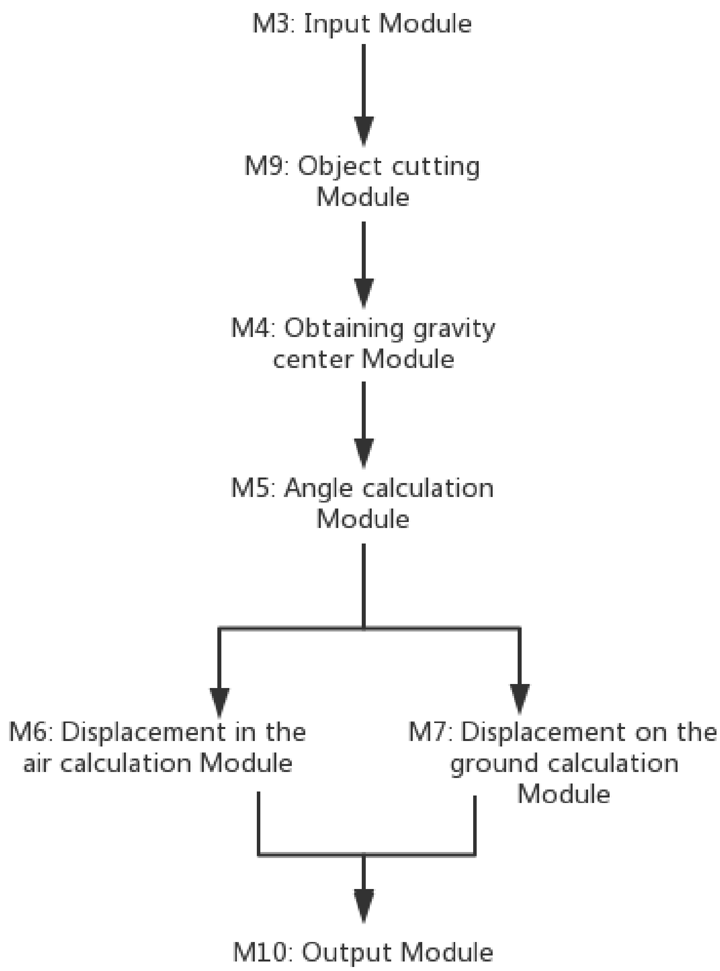

Figure 1: Use hierarchy among modules

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.