SE 3XA3: Software Requirements Specification
PineSweeper

Team 7, PineApple
Prince Sandhu; sandhps2
Varun Rathore; rathorvs
Vishesh Gulatee; gulatev

December 8, 2016

Contents

List of Tables

List of Figures

Table 1: **Revision History**

Date	**Version**	**Notes**
November 08	1.0	Created Initial LaTex File
November 08	1.1	Anticipated and Unlikely Changes Initial
November 10	1.2	Module Hierarchy Initial
November 10	1.3	Module Decomposition Initial
November 11	1.4	Traceability Matrix Initial
November 12	2.0	Anticipated and Unlikely Changes Final
November 12	2.1	Module Hierarchy Final
November 12	2.2	Module Decomposition Final
November 13	2.3	Traceability Matrix Final

1 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 2 lists the anticipated and unlikely changes of the software requirements. Section 3 summarizes the module decomposition that was constructed according to the likely changes. Section 4 specifies the connections between the software requirements and the modules. Section 5 gives a detailed description of the modules. Section 6 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 7 describes the use relation between modules.

2 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 2.1, and unlikely changes are listed in Section 2.2.

2.1 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: Different visual themes.

AC4: Different shape of the mine grid.

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

UC3: Clicking on a button will always uncover the the game item that is situated under the button.

UC4: Clicking on a *Pineapple* will always result in the game state being terminated.

UC5: Clicking on the reset button will always reset the game.

UC6: The game will always be played on a grid.

3 Module Hierarchy

This section provides an overview of the module design. Modules are summarized in a hierarchy decomposed by secrets in Table 2. The modules listed below, which are leaves in the hierarchy tree, are the modules that will actually be implemented.

M1: Input Format Module

M2: Game Mode Selection Module

M3: Preferences Selection Module

M4: Shared Data Module

M5: State Update Module

M6: Output Module

M7: Cell Data Structure Module

M8: Grid Generation Module

M9: Grid Traversal Module

M10: Application Objective Module

M11: Hardware Hiding Module

Level 1	**Level 2**
Hardware-Hiding Module	
Behaviour-Hiding Module	Input Format Module Game Mode Selection Module Preferences Selection Module Shared Data Module State Update Module Output Module
Software Decision Module	Cell Data Structure Module Grid Generation Module Grid Traversal Module Application Objective Module

Table 2: Module Hierarchy

4 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 3.

5 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.

5.1 Hardware Hiding Modules (M11)

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 and Java Virtual Machine

5.2 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: –

5.2.1 Input Format Module (M1)

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: *PineSweeper*

5.2.2 Game Mode Selection Module (M2)

Secrets: textcolorredGrid and Window Size.

Services: Allows user to select the difficulty of the game that they wish to play.

Implemented By: *PineSweeper*

5.2.3 Preferences Selection Module (M3)

Secrets: User preference.

Services: Allows user to select the visual theme (provided), for the application to embody.

Implemented By: *Java Libraries and OS*

5.2.4 Shared Data Module (M4)

Secrets: Difficulty level and theme choice.

Services: Stores and shares resources needed across all modules.

Implemented By: *PineSweeper and Java Libraries*

5.2.5 State Update Module (M5)

Secrets: Grid data.

Services: Updates and manipulates state variables and methods based on user response. Provides A.I. of the application.

Implemented By: *PineSweeper*

5.2.6 Output Module (M6)

Secrets: Application window.

Services: Updates application response to user input, by providing a graphical representation of the game state.

Implemented By: *PineSweeper and Java Libraries*

5.2.7 Application Objective Module (M10)

Secrets: Model

Services:

Implemented By: *PineSweeper*

5.3 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: –

5.3.1 Cell Data Structure Module (M7)

Secrets: Cells

Services: Stores Cell objects, that represent the buttons of the Grid in the application.

Implemented By: *PineSweeper and Java Libraries*

5.3.2 Grid Generation Module (M8)

Secrets: Grid

Services: Algorithm that describes generation of grid containing cells and its contents.

Implemented By: *PineSweeper*

5.3.3 Grid Traversal Module (M9)

Secrets: Grid

Services: Graph search algorithm that describes the nature with which the grid is traversed by the application.

Implemented By: *PineSweeper*

6 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	M2, M5
R2	M8, M7
R3	M5, M6
R4	M9, M8
R5	M5, M10
R6	M8, M9
R7	M5, M6
R8	M5, M6
R9	M4, M5, M6
R10	M6
R11	M11, M6
R12	M1, M2, M6, M8
R13	M2, M3
R14	M3, M6, M11
R15	M5, M6, M11
R16	M11
R17	M5, M6, M11
R18	M11
R19	M11
R20	M5, M6, M8

Table 3: Trace Between Requirements and Modules

AC	**Modules**
AC1	M11
AC2	M1
AC3	M6
AC4	M6

Table 4: Trace Between Anticipated Changes and Modules

7 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.

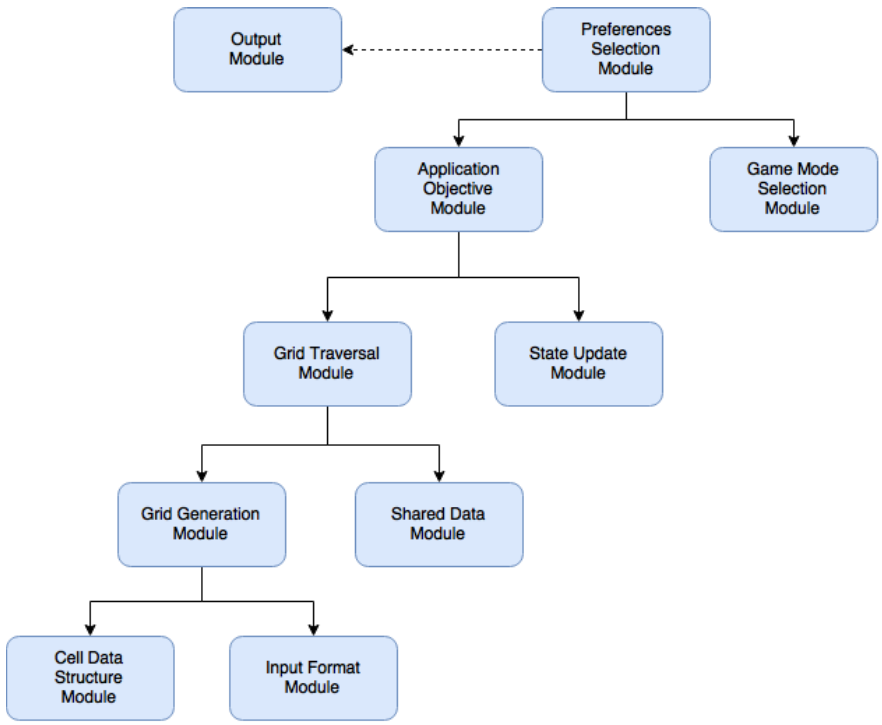

Figure 1: Use hierarchy among modules

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.

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.