

SE 3XA3: Software Requirements Specification Snake 2.0

Team 30, VUA30
Andy Hameed — 400073469
Student 2 name and macid
Student 3 name and macid

November 8, 2018

Contents

1 Introduction 1

2 Anticipated and Unlikely Changes 1

2.1 Anticipated Changes 2

2.2 Unlikely Changes 2

3 Module Hierarchy 2

4 Connection Between Requirements and Design 2

5 Module Decomposition 3

5.1 Hardware Hiding Modules (M1) 3

5.2 Behaviour-Hiding Module 3

5.2.1 Input Format Module (M??) 4

5.2.2 Etc. 4

5.3 Software Decision Module 4

5.3.1 Etc. 4

6 Traceability Matrix 4

7 Use Hierarchy Between Modules 5

List of Tables

1 Revision History i

2 Module Hierarchy 3

3 Trace Between Requirements and Modules 5

4 Trace Between Anticipated Changes and Modules 5

List of Figures

1 Use hierarchy among modules 6

Table 1: Revision History

Date	Version	Notes
Date 1	1.0	Notes
Date 2	1.1	Notes

1 Introduction

Almost everyone nowadays relies on a computer as a multipurpose tool for research, video streaming, gaming and many other tasks. With the emergence of fast computing, gaming has become a popular pastime activity and a source of entertainment for many. However, not everyone has a device powerful enough to support extensive game applications. A simple, memory-efficient application of the Snake game allows it to be accessible for gamers without the need for extensive hardware or a high-performance computer. Our team, VUA30, will be creating a desktop application for the well-known “Snake” game with new enhancements and features. This competitive and addictive game can allow the user to play at their own pace and challenge their own high score.

Buying a computing device with high storage and faster performance can be out of budget. Complicated software covers up all the storage and the user is bound to use these applications as opposed to downloading other software. The importance of the redevelopment of “The Snake” is to save computing device’s personal storage and allow the user to play a game 24/7 with strong performance, even offline. Creating a desktop version of the snake game can fit into the category of downloadable calssical games such as the solitaire suite. The recreation of this game will allow the user to enjoy the classical game anytime and anywhere as long as they have installed the application. Improving aspects such as graphics and custom speed will also make the game more interesting. We would like to add more features to the game to make it more customizable and help people enjoy the classical game in an exciting and new way.

The scope of the project will cover the reimplementaion of the game as well as enhancements to the game, in particular, changes that pertain to gameplay themes, different modes/difficulties and so on. Due to the time constraints of the project, not all enhancements will be implemented but at least one enhancement will be implemented to differentiate Snake 2.0 from previous remakes of the game.

The module guide document will outline the structure of the reimplementaion through modular programming. It highlights the components of the system such that each component can easily be identified by project members, whether they are maintaining or designing the software. Among the list of internal stakeholders that may make use of this document, any new members that are added to the team will also be able to use the Module Guide (MG) as a convenient reference to specific modules that they are concerned with or working on (Parnas et al., 1984).

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.

...

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.

...

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: Hardware-Hiding Module

...

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.

Level 1	Level 2
Hardware-Hiding Module	
	?
	?
	?
Behaviour-Hiding Module	?
	?
	?
	?
	?
	?
Software Decision Module	?
	?

Table 2: Module Hierarchy

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 (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

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 (M??)

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: [Your Program Name Here]

5.2.2 Etc.

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 Etc.

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	M1, M??, M??, M??
R2	M??, M??
R3	M??
R4	M??, M??
R5	M??, M??, M??, M??, M??, M??
R6	M??, M??, M??, M??, M??, M??
R7	M??, M??, M??, M??, M??
R8	M??, M??, M??, M??, M??
R9	M??
R10	M??, M??, M??
R11	M??, M??, M??, M??

Table 3: Trace Between Requirements and Modules

AC	Modules
AC1	M1
AC2	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??

Table 4: Trace Between Anticipated Changes and Modules

7 Use Hierarchy Between Modules

In this section, the uses hierarchy between modules is provided. ? 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

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.