SE 3XA3: Software Requirements Specification
Sudoku Solver

Team 08, SudoCrew
Rashad A. Bhuiyan (bhuiyr2)
Kai Zhu (zhuk2)
Stanley Chan (chans67)

March 17, 2022

Contents

1 Introduction **1**
- 1.1 Overview 1
- 1.2 Context 1
- 1.3 Design Principles 1
- 1.4 Document Structure 2

2 Anticipated and Unlikely Changes **3**
- 2.1 Anticipated Changes 3
- 2.2 Unlikely Changes 3

3 Module Hierarchy **3**

4 Connection Between Requirements and Design **4**

5 Module Decomposition **4**
- 5.1 Hardware Hiding Modules (M1) 5
- 5.2 Behaviour-Hiding Module 5
  - 5.2.1 CV Classification Module (M4) 5
  - 5.2.2 CV Results Module (M5) 5
  - 5.2.3 CV Errors Module (M6) 6
  - 5.2.4 Flask App Module (M7) 6
  - 5.2.5 Frontend Module (M9) 6
  - 5.2.6 Etc. 6
- 5.3 Software Decision Module 6
  - 5.3.1 Etc. 6

6 Traceability Matrix **6**

7 Use Hierarchy Between Modules **7**

List of Tables

1 **Revision History** ii
2 Module Hierarchy 4
3 Trace Between Requirements and Modules 7
4 Trace Between Anticipated Changes and Modules 7

List of Figures

1 Use hierarchy among modules 8

Table 1: **Revision History**

Date	**Version**	**Notes**
2022-03-16	0.0	Created Module Guide
2022-03-17	0.1	Finished Sections 1,
2022-03-18	0.2	...

1 Introduction

1.1 Overview

The purpose of this software application is to provide a comprehensive suite of tools for generating, recognizing, and solving Sudoku puzzles. The application will provide a web-based front-end with an intuitive interface to cater to users of different technical abilities on most modern hardware. Computer vision is also utilized to improve ease of use, by directly interfacing puzzles from print-media to the application.

1.2 Context

After completing the first stage of the design, the Software Requirements Specification (SRS), the Module Guide (MG) is developed (?). 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:

- 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.

1.3 Design Principles

The Design Principles used as the basis for the decomposition of the system into modules are the Information Hiding and Encapsulation principle, as well as the principles that a Uses Relation Hierarchy should have no cycle, have low coupling, and have high cohesion.

The Information Hiding principle states that each module hides a specified secret—related to a design decision—from the rest of the system. The Encapsulation principle states that any changeable information about a module is within its implementation details, but the module interface remains unchanged regardless of any changes to the implementation details. Cycles within a Uses Hierarchy indicates poor design as it implies the existence of infinite loops which indicates a need for further decomposition of modules into smaller components. Low coupling indicates that each module is not tightly dependent on other modules and that it can independently function without much use of other modules. High cohesion indicates that the elements within the module are strongly related to each other.

1.4 Document Structure

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 hosting format of the web application.

AC2: The resolution of the device that the web application will be running on (mobile, desktop, etc.)

2.2 Unlikely Changes

UC1: The rules to Sudoku.

UC2: Input/Output devices (Input: File and/or Keyboard, Output: File, Memory, and/or Screen).

UC3: There will always be a source of input data external to the source.

UC4: The purpose of the system is to allow users to play Sudoku as well as provide solutions to their Sudoku boards.

UC5: The system will not store any given files.

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

M2: Generation Module

M3: Solver Module

M4: CV Classification Module

M5: CV Results Module

M6: CV Errors Module

M7: Flask App Module

M8: Sudoku Utilities Module

M9: Frontend Module

Level 1	**Level 2**
Hardware-Hiding Module	
Behaviour-Hiding Module	CV Classification Module CV Results Module CV Errors Module Flask App Module Frontend Module
Software Decision Module	Generation Module Solver Module Sudoku Utilities 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 **?**. 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 CV Classification Module (M4)

Secrets: The algorithms and machine learning model that is used to interpret Sudoku boards.

Services: Interprets the inputted image and serializes the Sudoku board into an array format.

Implemented By: [Your Program Name Here]

5.2.2 CV Results Module (M5)

Secrets:

Services:

Implemented By: [Your Program Name Here]

5.2.3 CV Errors Module (M6)

Secrets:

Services:

Implemented By: [Your Program Name Here]

5.2.4 Flask App Module (M7)

Secrets: Route handling and Sudoku board handling logic.

Services: Serves as the backend for the web application and handles different routes and the high level logic of the application.

Implemented By: [Your Program Name Here]

5.2.5 Frontend Module (M9)

Secrets: Rendering details of the web application.

Services: Acts as the interface for the user to interact with the web application.

Implemented By: [Your Program Name Here]

5.2.6 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**
AC??	M1
AC??	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