SE 3XA3: Software Requirements Specification
Random Flag Generator

Team #2, Team Jakriel
Akram Hannoufa, hannoufa
Ganghoon (James) Park, parkg10
Nathaniel Hu, hun4

March 18, 2022

Contents

List of Tables

List of Figures

Table 1: **Revision History**

Date	**Version**	**Notes**
March 15, 2022	1.0	Initial Document
March 16, 2022	1.1	Added initial drafts of Module Hierarchy and Module Decomposition Sections
March 17, 2022	1.2	Added initial draft of Anticipated and Unlikely Changes Section; added uses hierarchy diagram to Use Hierarchy Between Modules Section; added traceability matrices to Traceability Matrix Section
March 17, 2022	1.3	Added and modified Intro and Connection sections.

1 Introduction

Flag Generator is an implementation of the open-source project PAGAN, Python Avatar Generation for Absolute Nerds. The main functionality of the program comes from generating a hashcode from a user's input string and creating a unique graphical output based on the generated hashcode. Flag Generator generates a unique flag with its own set of colours, symbols, designs, etc. all based on the hashcode generated from the user's input.

1.1 Purpose

The purpose of the Module Guide document is to outline how this system is modularly decomposed. Following a module decomposition approach, the information hiding principle is being adhered to (Parnas, 1972). Following this principle also allows new project members, maintainers, and designers to be able to easily identify the individual components of the software, derive an understanding of the hierarchical structure of the system, and to easily verify certain software criteria such as consistency, flexibility, and feasibility. Additionally, by breaking up the system into modules, the principles of low cohesion and high coupling.

1.2 Scope

The modules used to develop this program were based off of the requirements outlined in the Software Requirements Specification. The modules' external behaviours are explicitly described and outlined in the corresponding Module Interface Specification document. Through adhering to the principle of information hiding, anticipated and unlikely changes are represented by the secrets of the modules, allowing for easier changes to be made in the future (design for change). Lastly, a uses hierarchy is established to grpahically show the relationships between the modules.

1.3 Acronyms, Abbreviations, and Symbols

Table 2: **Table of Abbreviations**

Abbreviation	**Definition**
GUI	Graphical User Interface
UI	User Interface
FR	Functional Requirement
RGB	Red Green Blue
SRS	Software Requirements Specification
PAGAN	**Python Avatar Generator for Absolute Nerds**

Table 3: **Table of Definitions**

Term	**Definition**
Python	The programming language used in this project.
Pytest	Python testing tool that allows testers to write test code and create simple and scalable test cases.
Input String	The input of type string from the user.
Gallery	Collection of previously generated flags.
Hashing	Algorithm that converts input data to a fixed-size value. A hashing function usually outputs a string or hexadecimal value.
User	Person who uses or operates a computer program.
User Interface	Where interactions between machines and humans occur.
Graphical User Interface	A form of UI that allows users to use electronic devices using interactive graphics.
System/Program	Collection of instructions or components that tell a computer how to operate.
Software Requirements Specification	A document that details what the program/software will do and how it will accomplish the expected performance/tasks.
Tester	An individual testing the software via the user interface or the code/test cases.
Typeform	Website that is a software as a service that specializes in creating and building online surveys.

1.4 Overview

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 addition of new flag assets.

AC2: The flag resolution options.

AC3: Available hashing functions.

AC4: Maximum size of flag gallery (number of images).

AC5: Where and how flags are displayed after being generated.

AC6: More user control over user input streams; output.

AC7: Output file format.

AC8: Input string criteria, wider range of allowable characters (broader input range).

AC9: Making changes to how to use the program.

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: The same graphics library (PIL/pillow) will be used to generate the flag image files

UC2: The same asset reader (JKAReader) will be used to read the flag symbol design.

UC3: String input

UC4: Input/output devices (input: keyboard, output: monitor and file)

UC5: Hashing function

3 Module Hierarchy

This section provides an overview of the module design. Modules are summarized in a hierarchy decomposed by secrets in Table 4. The modules listed below, which are leaves in the hierarchy tree, are the modules that will actually be implemented.

M1: HashGenerator Module

M2: FlagAssetsLib Module

M3: JKAReader Module

M4: DecisionUtilities Module

M5: HashToFlag Module

M6: FlagGenerator Module

M7: GUI Module

M8: Display Module

M9: Gallery Module

M10: HelpInstructions Module

M11: Settings Module

Level 1	**Level 2**
Hardware-Hiding Module	
Behaviour-Hiding Module	HashGenerator FlagAssetsLib JKAReader GUI Display Gallery HelpInstructions Settings
Software Decision Module	DecisionUtilities HashToFlag FlagGenerator

Table 4: 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 5, in the form of a traceability matrix.

Every functional requirement is implemented by at least one module, and similarly the modules responsible for handling the anticipated changes are also laid out in a traceability matrix, Table 6.

Modules are generally broken up into hash generation, hashcode to flag decisions and GUI modules. Within these categories, there is a breakdown of modules to handle specific functionalities. The GUI functionality is broken down into Display, Gallery, Help, and Settings modules to handle specific funcional requirements. Similarly, HashToFlag's functionality is broken down into a FlagAssetsLib, HashGenerator (module responsible for generating the hashcode), and a DecisionUtilities module (helper functions for converting a hashcode to decisions). Lastly, the JKAReader module (responsible for converting .jka files into pixels) combined with the HashToFlag module comprises the majority of the FlagGenerator functionality.

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.

5.1 Hardware Hiding Modules

N/A

5.2 Behaviour-Hiding Module

5.2.1 HashGenerator Module (M1)

Secrets: Hashing algorithms available and method for generating hash digests from input strings and selected hashing algorithm (if available)

Services: Generates hash digests from input strings using selected hashing algorithms (if available; default algorithm is SHA-256)

Implemented By: Random Flag Generator

5.2.2 FlagAssetsLib Module (M2)

Secrets: Flag assets, constants and values used to generate flags

Services: Provides the data files and values needed to generate flags

Implemented By: Random Flag Generator

5.2.3 JKAReader Module (M3)

Secrets: Method for parsing flag asset (.jka) file data into usable form for generating flags

Services: Parses flag asset (.jka) file data into pixel maps to be used to generate flags

Implemented By: Random Flag Generator

5.2.4 GUI Module (M4)

Secrets: Methods for starting the program and presenting the GUI view, taking in user input strings and selected settings for generating flags

Services: Starts the program and presents the GUI view, takes in user input strings and selected settings for generating flags

Implemented By: Random Flag Generator

5.2.5 Display Module (M5)

Secrets: Methods for displaying the associated generated flags

Services: Displays the associated generated flags via a GUI

Implemented By: Random Flag Generator

5.2.6 Gallery Module (M6)

Secrets: Saved flags in user's flag gallery and methods for displaying all saved flags from the user's flag gallery and searching up flags by name (i.e. input string)

Services: Displays all the saved flags from the user's flag gallery via an interactive GUI

Implemented By: Random Flag Generator

5.2.7 HelpInstructions Module (M7)

Secrets: Help and instructions information and methods for displaying the help and instructions menu

Services: Displays the helps and instructions menu and information via a GUI

Implemented By: Random Flag Generator

5.2.8 Settings Module (M8)

Secrets: Program version and methods for displaying the settings menu, taking in user settings changes and updating settings

Services: Displays the settings menu and program version via an interactive GUI

Implemented By: Random Flag Generator

5.3 Software Decision Module

5.3.1 DecisionUtilities Module (M9)

Secrets: Auxiliary helper methods for padding generated hash digests (that are too short) and assisting in flag generation information parsing and usage in generating flags, and associated constants and values that will be used by the auxiliary helper methods

Services: Pads generated hash digests (that are too short) and assists in flag generation information parsing and usage in generating flags

Implemented By: Random Flag Generator

5.3.2 HashToFlag Module (M10)

Secrets: Methods for grinding generated hash digests to select the colours and flag design elements to generate flags

Services: Provides the selected colours and flag design elements obtained from grinding the generated hash digest to generate flags

Implemented By: Random Flag Generator

5.3.3 FlagGenerator Module (M11)

Secrets: Methods for obtaining the colours and flag design elements and generating the flags and saving the flag image files to the flag gallery

Services: Obtains the colours and flag design elements and generates the flags and saves the flag image files to the flag gallery

Implemented By: Random Flag Generator

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
FR1	M7
FR2	M7
FR3	M7
FR4	M1, M2, M3, M4, M5, M6, M7
FR5	M8
FR6	M1, M2, M3, M4, M5, M6, M11
FR7	M6
FR8	M1, M2, M3, M4, M5, M6, M7
FR9	M8
FR10	M1, M2, M3, M4, M5, M6, M11
FR11	M6, M7, M9
FR12	M10
FR13	M10
FR14	M11
FR15	M11
FR16	M11
FR17	M9
FR18	M9
FR19	M9

Table 5: Trace Between Requirements and Modules

AC	Modules
AC1	M2, M3
AC2	M2
AC3	M1
AC4	M9
AC5	M8
AC6	M4, M5, M6, M7, M11
AC7	M6
AC8	M1, M7
AC9	M10

Table 6: 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.

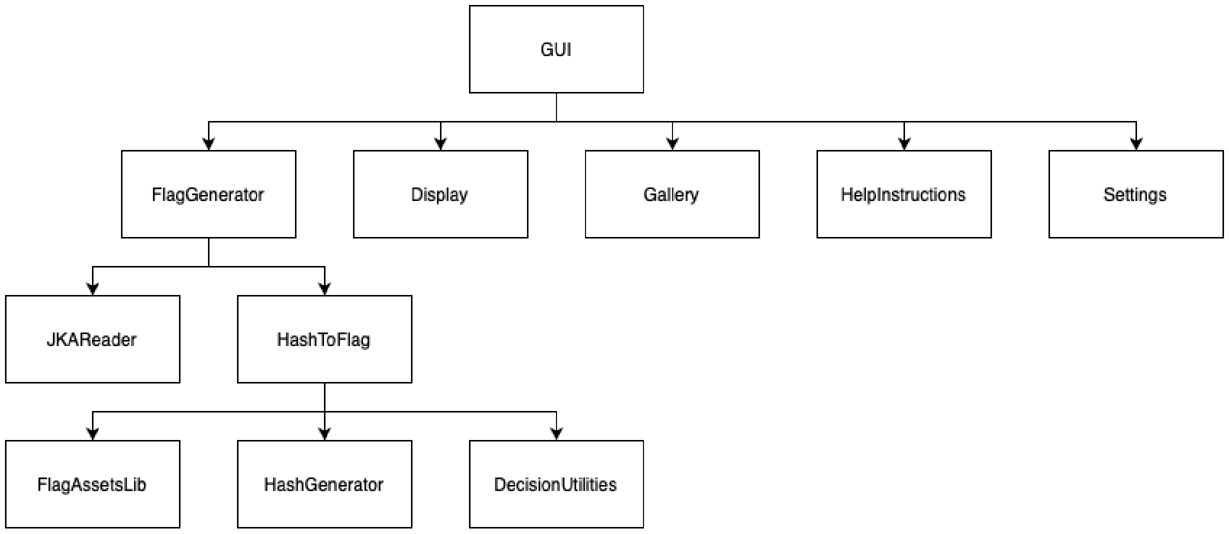

Figure 1: Use hierarchy among modules

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. 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.