# SE 3XA3: Module Guide

Group #10
Lab: L03
Aamina Hussain, hussaa54
Jessica Dawson, dawsor1
Fady Morcos, morcof2

# Contents

**1 Introduction** **1**

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

**3 Module Hierarchy** **2**

**4 Connection Between Requirements and Design** **4**

**5 Module Decomposition** **4**
- 5.1 Hardware Hiding Modules (M1) . . . 4
- 5.2 Behaviour-Hiding Module . . . 4
  - 5.2.1 Controller UI Module (M2) . . . 5
- 5.3 Software Decision Module . . . 5
  - 5.3.1 Canvas Module (M3) . . . 5
  - 5.3.2 Widget Module (M4) . . . 5
  - 5.3.3 Layer Module (M5) . . . 5
  - 5.3.4 Overlay Module (M6) . . . 6
  - 5.3.5 Color Palette Module (M7) . . . 6
  - 5.3.6 Help Module (M8) . . . 6
  - 5.3.7 Generate Shape Module (M9) . . . 6
  - 5.3.8 Widget Storage Module (M10) . . . 6
  - 5.3.9 Assets Module (M11) . . . 7
  - 5.3.10 Text Overlay Module (M12) . . . 7
  - 5.3.11 Switch Theme Module (M13) . . . 7

**6 Traceability Matrix** **7**

**7 Use Hierarchy Between Modules** **8**

# List of Tables

1 **Revision History** . . . 1
2 Module Hierarchy . . . 3
3 Trace Between Requirements and Modules . . . 8
4 Trace Between Anticipated Changes and Modules . . . 8

# List of Figures

1 Use hierarchy among modules . . . 9

Table 1: **Revision History**

| **Date** | **Version** | **Notes** |
|---|---|---|
| March 16, 2022 | 0.0 | Initial Document; Completed Section 3, 7 |
| March 17, 2022 | 0.1 | Completed Section 2, 6 |
| March 18, 2022 | 0.2 | Completed Section 1, 4, 5 |
| April 11, 2022 | 1.0 | Added modules (M12, M13), updated Traceability Matrix/Uses Hierarchy accordingly |

## 1 Introduction

This document is the module guide for the Abstract Art Generator program. This document is intended to be read by:

- Designers and Developers
- Future project members/maintainers
- Stakeholders

The *Abstract Art Generator* project is a Python program which uses Pygame and Pygame _GUI to display a graphical user interface that allows interaction with the user. It generates random abstract art images that may be exported as PNG files.

Most successful systems are composed of multiple modules, and this modularization allows for better understandability and maintainability. It also creates a simple way to add new features and deal with the anticipated changes without having to re-implement the entire system.

The Python Painters Team made the decision to follow a *Design for Change* design pattern, as well as the concept of information hiding, where each module holds a single secret.

This document will break down the system into modules, each of which will have their own MIS specified in the MIS document. It will also contain a traceability matrix mapping every requirement from the SRS document to at least one of the modules.

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 ways layers can be drawn, new or modified drawing algorithms.

**AC2:** What widgets exist.

**AC3:** The layout of the UI.

**AC4:** The contents of the help dialogue.

**AC5:** The list of color palettes.

**AC6:** The list of fonts.

### 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 library used for input and graphical display.

**UC2:** The existence of the color palette widget.

## 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:** Controller UI Module

**M3:** Canvas Module

**M4:** Widget Module

**M5:** Layer Module

**M6:** Overlay Module

**M7:** Color Palette Module

**M8:** Help Module

**M9:** Generate Shape Module

**M10:** Widget Storage Module

**M11:** Assets Module

**M12:** Text Overlay Module

**M13:** Switch Theme Module

| **Level 1** | **Level 2** |
|---|---|
| Hardware-Hiding Module | Pygame Library |
| Behaviour-Hiding Module | Controller UI Module |
| Software Decision Module | Canvas Module<br>Widget Module<br>Layer Module<br>Overlay Module<br>Color Palette Module<br>Help Module<br>Generate Shape Module<br>Widget Storage Module<br>Assets Module<br>Text Overlay Module<br>Switch Theme 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 which is located in section 7 of this document. However readers of this document are recommended to read section 3 and section 5 first before looking at the traceability matrix in section 7.

## 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:** Pygame

### 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:** N/A

### 5.2.1 Controller UI Module (M2)

**Secrets:** The processes required to convert user input into a display.

**Services:** Takes in input data from the user and displays corresponding visual output.

**Implemented By:** AbstractArtGenerator

## 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:** N/A

### 5.3.1 Canvas Module (M3)

**Secrets:** The decision making for how the pygame surfaces are placed in the display.

**Services:** Includes the main decision making for how the art image is produced.

**Implemented By:** AbstractArtGenerator

### 5.3.2 Widget Module (M4)

**Secrets:** The abstract methods other classes can inherit.

**Services:** An abstract class that contains a framework of methods other classes can inherit and use. This accounts for anticipated changes.

**Implemented By:** AbstractArtGenerator

### 5.3.3 Layer Module (M5)

**Secrets:** The processes required to generate a layer.

**Services:** Contains all methods relating to generating, displaying, and interacting with the layers of the generated art image.

**Implemented By:** AbstractArtGenerator

#### 5.3.4 Overlay Module (M6)

**Secrets:** The processes required to use an overlay/border.

**Services:** Contains all methods relating to displaying and interacting with the overlays/borders that are placed over top of the generated art image.

**Implemented By:** AbstractArtGenerator

#### 5.3.5 Color Palette Module (M7)

**Secrets:** The processes required to use the color palettes.

**Services:** Contains all methods relating to displaying, randomly choosing, and interacting with the color palette options.

**Implemented By:** AbstractArtGenerator

#### 5.3.6 Help Module (M8)

**Secrets:** How the instructions are displayed.

**Services:** Contains all methods relating to displaying the instructions and interacting with the help option.

**Implemented By:** AbstractArtGenerator

#### 5.3.7 Generate Shape Module (M9)

**Secrets:** The processes required to generate shapes.

**Services:** Contains methods that are used to generate different shapes that are a part of each layer of the art image.

**Implemented By:** AbstractArtGenerator

#### 5.3.8 Widget Storage Module (M10)

**Secrets:** Storage for program widgets.

**Services:** Allows all other modules to access `program widgets`. The widgets include the layer module, help module, overlay module, and color palette module.

**Implemented By:** AbstractArtGenerator

#### 5.3.9 Assets Module (M11)

**Secrets:** Storage for assets (such as constants).

**Services:** Contains assets that are used by other modules.

**Implemented By:** AbstractArtGenerator

#### 5.3.10 Text Overlay Module (M12)

**Secrets:** The processes required to add text on top of the generated art image.

**Services:** Contains all methods relating to creating, displaying, and interacting with text on top of the generated art image.

**Implemented By:** AbstractArtGenerator

#### 5.3.11 Switch Theme Module (M13)

**Secrets:** The processes required to change the UI theme.

**Services:** Contains all methods related to changing and interacting with the UI theme options.

**Implemented By:** AbstractArtGenerator

## 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 | M2, M7 |
| FR3 | M2, M3 |
| FR4 | M5, M9 |
| FR5 | M2, M5 |
| FR6 | M3, M12 |
| FR7 | M2, M3, M11, M12 |
| FR8 | M2, M3, M11, M12 |
| ~~FR9~~ | ~~M2, M3, M11~~ |
| FR9 | M2, M3, M6 |
| FR10 | M2, M6 |
| FR11 | M2, M8, M11 |
| FR12 | M2, M13 |
| LF1 | M3, M5, M6, M7, M12 |
| LF2 | M2 |
| UH1 | M2 |
| UH2 | M2, M5, M6, M7, M8, M11, M12 |
| UH3 | M2, M8 |
| PR1 | M2 |
| PR2 | M3 |

Table 3: Trace Between Requirements and Modules

| AC | Modules |
|---|---|
| AC1 | M5, M6, M9 |
| AC2 | M2, M4, M10 |
| AC3 | M2 |
| AC4 | M8 |
| AC5 | M7 |
| AC6 | M12 |

Table 4: Trace Between Anticipated Changes and Modules

## 7 Use Hierarchy Between Modules

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

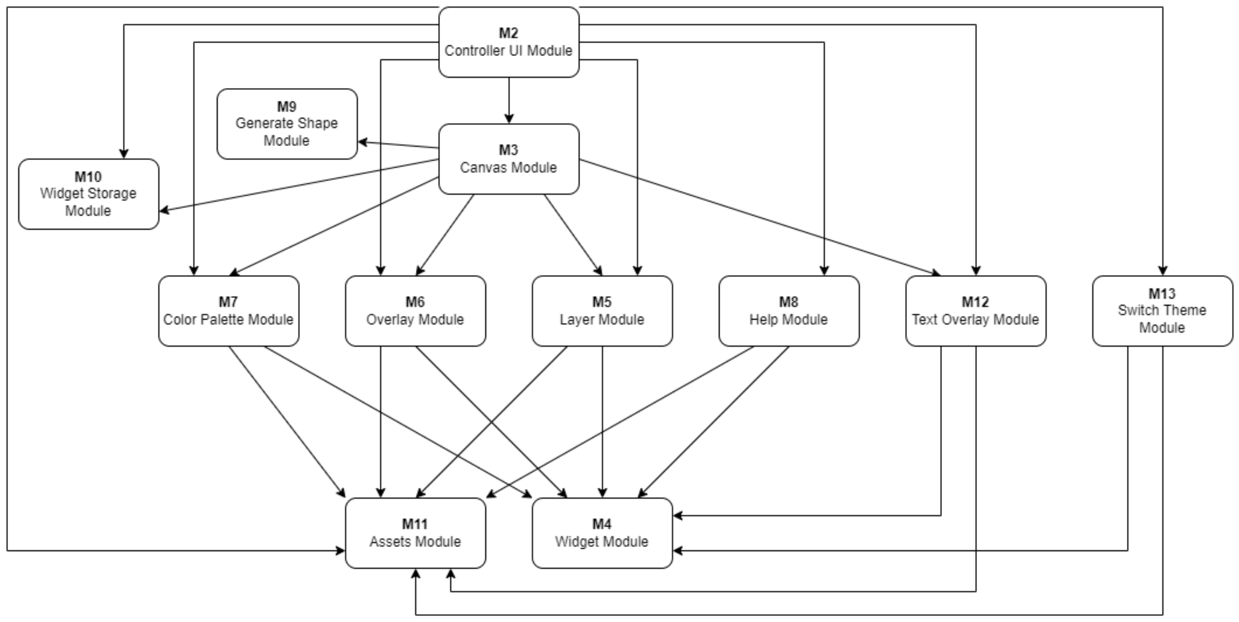

Figure 1: Use hierarchy among modules

## 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.