***Software Project Final Report***

Image Editor

Khizzer Ahmed, Abdullah Syed, Roland Vargas

12/5/2024

**Table of Contents**

**1 Introduction p.3**

**1.1 Purpose and Scope p.3**

**1.2 Product Overview p.3**

**1.3 Structure of the Document p.3**

**1.4 Terms, Acronyms, and Abbreviations p.3**

**2 Project Management Plan p.3**

**2.1 Project Organization p.3**

**2.2 Lifecycle Model Used p.3**

**2.3 Risk Analysis p.3**

**2.4 Hardware and Software Resource Requirements p.3**

**2.5 Deliverables and Schedule p.4**

**3 Requirement Specifications p.4**

**3.1 Stakeholders for the System p.4**

**3.2 Use Cases p.5**

**3.2.1 Graphical Use Case Model p.5**

**3.2.2 Textual Description for each Use Case p.5-7**

**3.3 Rationale for Use Case Model p.7-8**

**3.4 Non-Functional Requirements p.8-9**

**4 Architecture p.9**

**4.1 Architectural Style(s) Used p.9**

**4.2 Architectural Model p.9**

**4.3 Technology, Software, and Hardware Used p.9**

**4.4 Rationale for your Architectural Style and Model p.9**

**5 Design p.10**

**5.1 User Interface Design p.10**

**5.2 Components Design p.10**

**5.3 Rationale for your detailed design models p.10**

**5.4 Traceability from Requirements to Detailed Design Models p.10**

**6 Test Management p.11**

**6.1 Complete List of System Test Cases p.11**

**6.2 Traceability of Test Cases to Use Cases p.11**

**6.3 Techniques Used for Test Case Generation p.11**

**6.4 Test Results and Assessments p.12-13**

**6.5 Defects Reports p.13-14**

**7 Conclusions p.14**

**7.1 Outcomes of the Project p.14**

**7.2 Lessons Learned p.14**

**7.3 Future Development p.14**

List of Figures

**Figure 1:** Deliverables and Milestones (Section 2.5) **p.4**

**Figure 2:** Graphical Use Case Model for the Image Editor (3.2.1) **p.5**

**Figure 3:** Architectural Diagram of the System Components (Section 4.2) **p.9**

**Figure 4:** UI Mockup for the Main Editor Window (Section 5.1) **p.10**

List of Tables  
**Table 1:** Textual Description of each Use Cases (Section 3.2.2) **p.5-7**

**Table 2:** Non-Functional Requirements (Section 3.4) **p.8-9**

**Table 3:** Techniques Used for Test Case Generation (Section 6.3) **p.11**  
**Table 4:** Test Cases and Results Summary (Sections 6.4) **p.12-13**  
**Table 5:** Defect Reports and Resolution Status (Section 6.5) **p.13-14**

**1 Introduction**

**1.1 Purpose and Scope**

The purpose of this project is to practice software engineering principles and create a simple and intuitive image editor that allows users to create and edit images. The scope of the project includes implementing essential image editing features such as loading and saving images in various formats, filters, brush customization, layer management, and transparency support. Features may be scrapped if there is not enough time to implement them.

**1.2 Product Overview**

The image editor provides a comprehensive toolkit for creating and editing images. Users can load existing images, create new ones, and apply modifications such as brushstrokes, filters, and layer adjustments.

**1.3 Structure of the Document**

This document presents the key aspects of the project, including the management plan, requirements, architecture, design, testing processes, and final outcomes, to provide a holistic view of the system's development and implementation.

**1.4 Terms, Acronyms, and Abbreviations**

PNG: Portable Network Graphics, an image file format

UI: User Interface

IDE: Integrated Development Environment

**2 Project Management Plan**

**2.1 Project Organization**

The project team consists of three developers, each contributing to design, implementation, and testing. Tasks were distributed based on expertise to ensure timely completion.

**2.2 Lifecycle Model Used**

The project follows a waterfall model, progressing sequentially from requirements gathering to design, implementation, testing, and delivery.

**2.3 Risk Analysis**

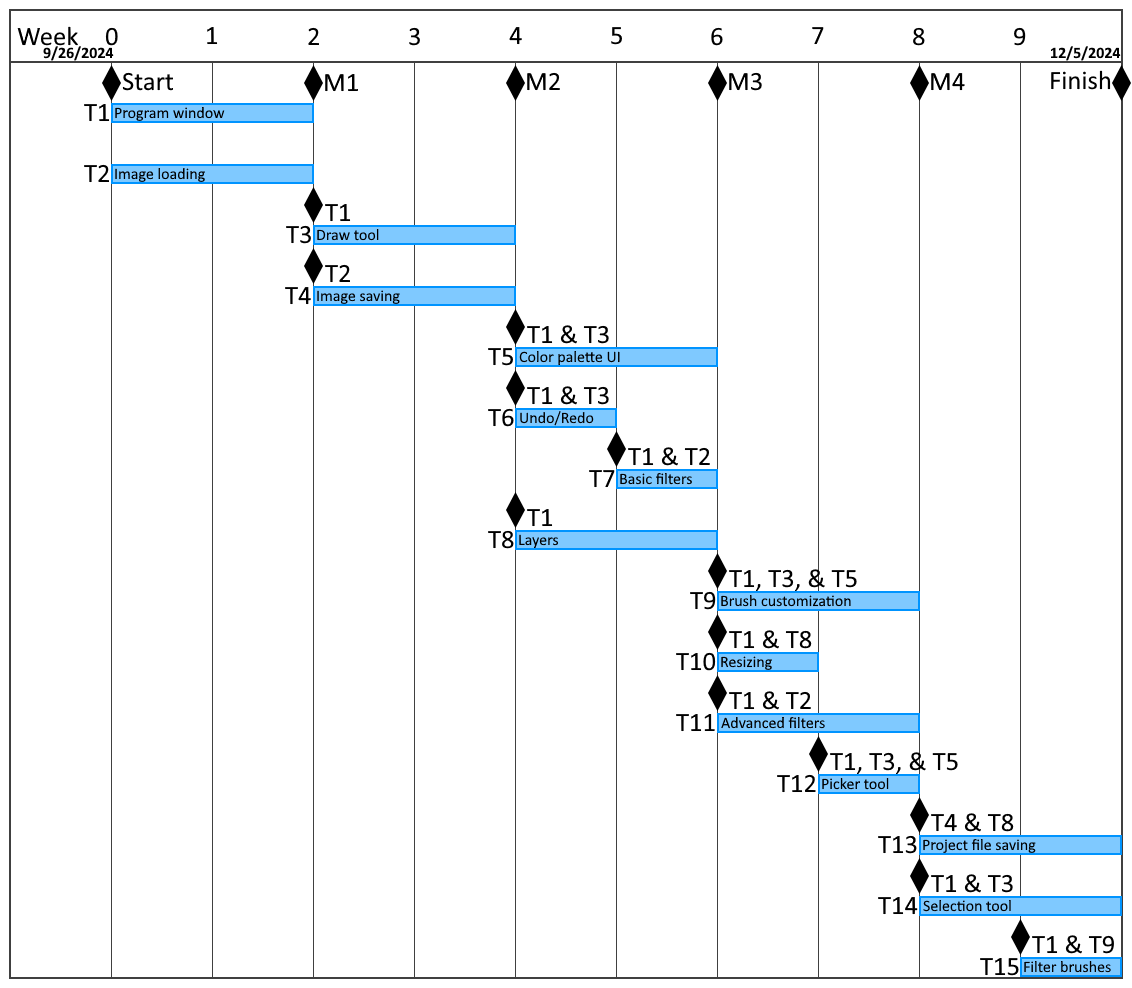
Key risks identified include delays in implementing features, difficulty integrating components, and knowledge gaps in certain areas. These were mitigated by prioritizing core features, scheduling regular progress reviews, and leveraging available resources to address challenges.

**2.4 Hardware and Software Resource Requirements**

The project requires a standard modern PC, recommended with Windows 10 or 11. Python is required, as well as the Pillow and Tkinter libraries to enable key functionalities.

**2.5 Deliverables and Schedule**

The primary deliverable is a fully functional image editor with essential features like image editing, filtering, and layering, delivered within the project timeline. The software and necessary documentation are to be completed by December 5th, 2024. Features may be scrapped to meet the deadline if development falls behind schedule.



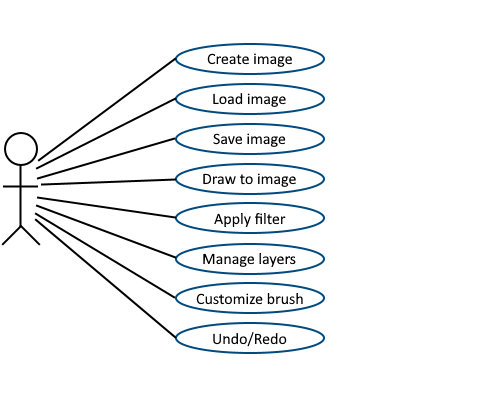
**3 Requirement Specifications**

**3.1 Stakeholders for the System**

The stakeholders include casual users seeking basic editing tools, creative individuals requiring advanced features like layers and filters, and anyone looking for an accessible and reliable image editor.

**3.2 Use Cases**

**3.2.1 Graphical Use Case Model**



**3.2.2 Textual Description for each Use Case**

| **Use Case** | **Description** | **Actors** | **Preconditions** | **Main Flow** | **Post Conditions** |
| --- | --- | --- | --- | --- | --- |
| Create Image | Allows the user to create a blank image with user-input dimensions | User | The application must run, and valid dimensions (at least 1 width and height) must be provided by the user. | 1. The user selects the "Create Image" option.  2. The user inputs the desired image dimensions. | A new blank image is created in the software. |
| Load Image | Allows the user to load an image file into the application. | User | The application must run, and the user must have an image file available. | 1. The user selects the "Load Image" option.  2. File dialog appears for the user to select an image.  3. The selected file is loaded and displayed on canvas. | The image is ready for editing. |
| Draw to Image | It enables the user to modify the image using various tools. | User | An image must be loaded or created. | 1. The user selects a tool (e.g., brush, eraser) from the toolbar.  2. The user interacts with the canvas to apply changes. | Changes are applied to the image. |
| Save Image | Saves the modified image to the user's desired location. | User | An image must be loaded or created and edited. | 1. The user selects the "Save Image" option.  2. File dialog appears for location and format.  3. The image is saved. | The image file is saved. |
| Manage Layers | Allows users to manage image layers (add, delete, reorder). | User | A project or image with layers must be loaded. | 1. The user selects the "Layers" panel.  2. The user performs actions like adding, deleting, or reordering layers. | The layers are updated. |
| Customize Brush | Enables the user to modify brush properties such as size, shape, and color. | User | The "Brush" tool must be selected. | 1. The user opens the brush customization panel.  2. User adjusts brush properties. | Changes apply to subsequent brush strokes. |
| Undo/Redo | Allows users to revert or reapply recent changes. | User | Changes must have been made to the image. | 1. The user selects the "Undo" or "Redo" button.  2. System reverts or reapplies changes. | The image reflects the chosen state. |
| Apply Filters | Applies filters to the entire image for enhancements or effects. | User | An image must be loaded. | 1. The user selects a filter from the toolbar or menu.  2. Filter is applied to the image. | The image is updated with the filter effect. |

**3.3 Rationale for Use Case Model**

1. **Clarity for Stakeholders:**
   * It simplifies understanding by visually representing interactions between users and the system.
2. **Modular Development:**
   * Each use case corresponds to a specific module, facilitating focused development and testing.
3. **Traceability:**
   * Use cases link directly to functional requirements, ensuring comprehensive coverage.
4. **Prioritization:**
   * By identifying key user actions (e.g., Load, Save, Edit), the team can focus on essential features while deferring less critical ones.
5. **Usability Focus:**
   * Highlighting actions like "Customize Brush" and "Manage Layers" ensures the system is designed with user needs in mind.

**3.4 Non-Functional Requirements**

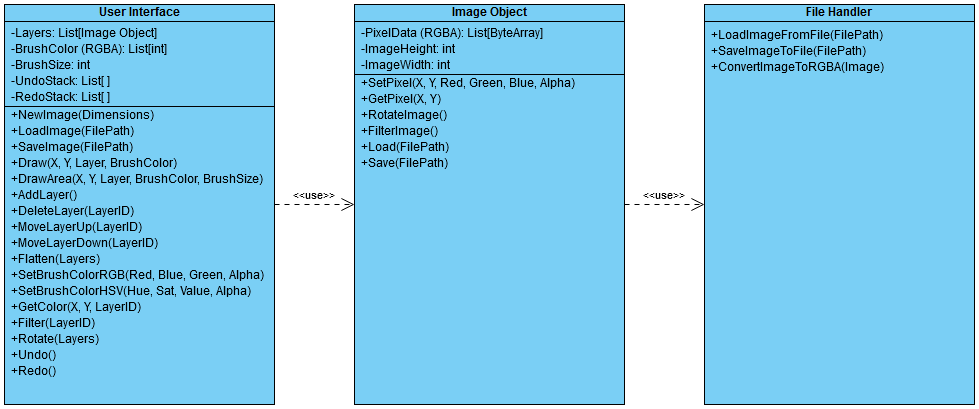
| **Category** | **Requirement** | **Details** |
| --- | --- | --- |
| Performance | Image Size Handling | The system must process images up to 1000x1000 pixels without noticeable lag. This is arbitrary because there isn’t a hard limit on possible image size, and we want to test a reasonably-sized image that isn’t likely to be too large to the point it will inevitably take a noticeably long time for every operation. Operations like drawing and simple filtering should be completed within 3 seconds. Sharpen and Blur filters are permitted to exceed this, but should take no more than 10 seconds. |
| Performance | Layer Management | The application should support at least 5 layers without performance degradation. |
| Usability | Intuitive User Interface | The UI should be easy to navigate, with clear labels and accessible tools (e.g., brush panel and layer controls within one or two clicks). |
| Usability | Undo/Redo Feedback | Users should receive a visual confirmation when undoing or redoing changes. |
| Compatibility | Operating System | The software must run on Windows 11 without additional setup beyond Python and required libraries. |
| Compatibility | File Formats | Must support saving/loading of PNG files with transparency; loading for JPEG and BMP formats. |
| Maintainability | Modular  Architecture | The software should use a modular design to facilitate adding or modifying features (e.g., new filters or tools). |
| Maintainability | Readable Code | The code should follow standard conventions with clear comments for future developers. |
| Tools and Licensing | Open-Source Tools | The project must use free libraries like Pillow and Tkinter to avoid licensing issues. |
| Error Handling | Error Handling | The system should display user-friendly error messages for issues like unsupported file formats or out-of-memory errors. |

**4 Architecture**

**4.1 Architectural Style(s) Used**

The system employs a modular architecture with three main components: the User Interface, Image Object, and File Handler. This ensures a clean separation of concerns and easy maintenance.

**4.2 Architectural Model**



**4.3 Technology, Software, and Hardware Used**

Programming Language**:** Python.

Libraries Used**:** Pillow for image file handling, Tkinter for graphical user interface design.

Hardware**:** The system is developed for modern PCs running Windows 11.

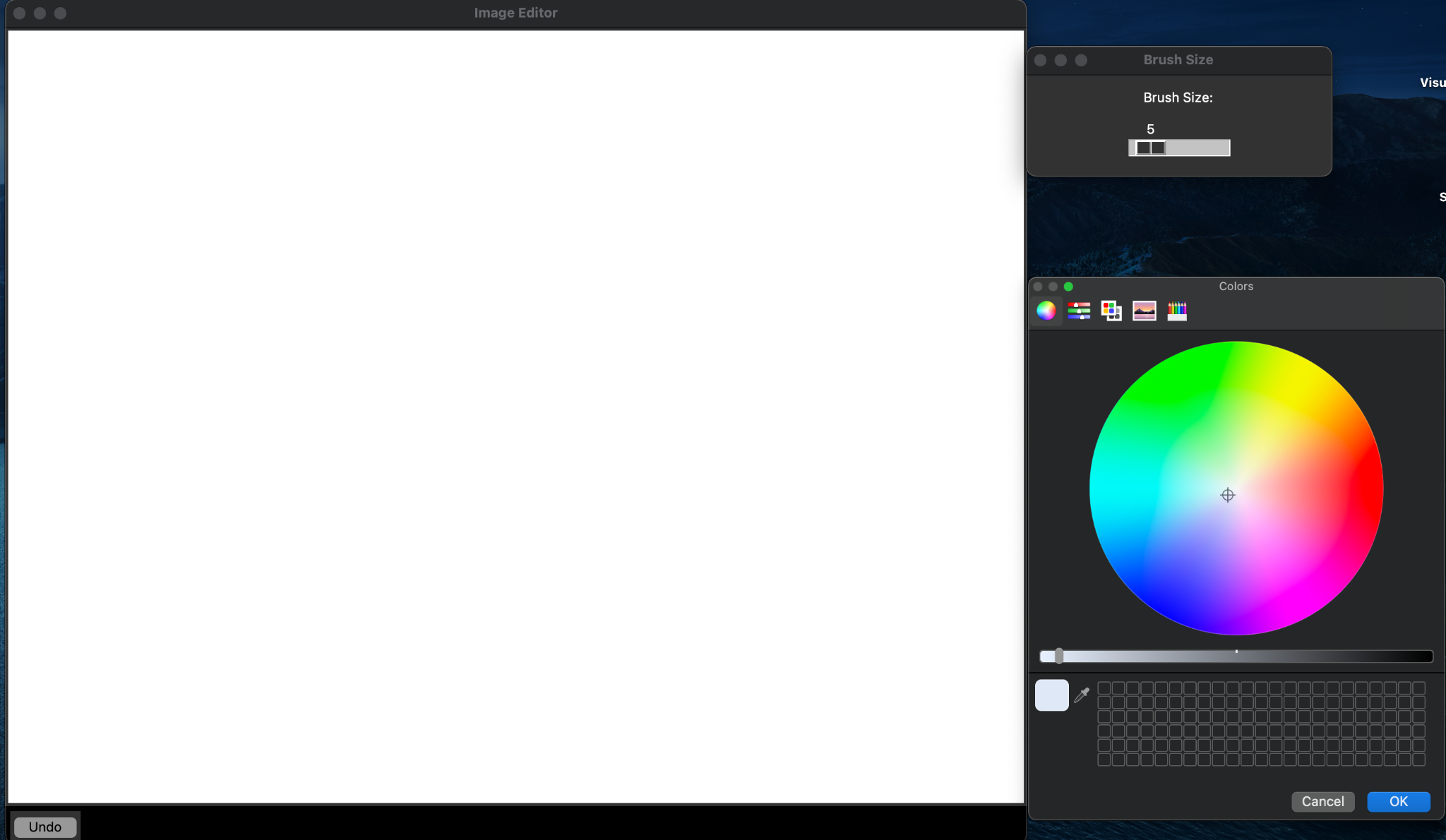
**4.4 Rationale for your Architectural Style and Model**

The modular design was chosen for its scalability and maintainability. By isolating key functionalities into separate components, the architecture ensures that each part can be developed, tested, and modified independently. This approach also simplifies debugging and enhances the ability to introduce new features in the future.

**5 Design**

**5.1 User Interface Design**

The user interface features a main window for image display, a toolbar for accessing tools and options, a color palette for customization, and panels for layer and brush settings. The design ensures intuitive navigation and ease of use.



**5.2 Components Design**

The system has three main components:

**User Interface:** Handles user input and displays outputs.

**Image Object:** Manages image data, layers, and tools like brushes and filters.

**File Handler:** Handles loading and saving images in supported formats.

When users interact with the UI (e.g., drawing or saving), commands are sent to the Image Object or File Handler, and results are reflected on the canvas.

**5.3 Rationale for your detailed design models**

The modular design keeps components separate, making the system easier to develop, test, and expand.

**5.4 Traceability from Requirements to Detailed Design Models**

Each requirement maps directly to a design element:

Loading/saving images: File Handler.

Drawing tools and filters: Image Object.

Layer management: Image Object and UI.

Undo/Redo: Image Object stacks.

**6 Test Management**

**6.1 Complete List of System Test Cases**

Test cases cover core functionalities such as:

Loading and saving images in various formats.

Drawing on the canvas using brushes.

Applying filters to images.

Managing layers (add, delete, reorder).

Undo and redo operations.

**6.2 Traceability of Test Cases to Use Cases**

Each test case corresponds to specific use cases:

Loading and saving tests verify the system's ability to handle files.

Drawing and brush tests align with user interaction use cases.

Layer management tests ensure all layer-related functionalities work.

Undo/redo tests validate navigation between editing states.

**6.3 Techniques Used for Test Case Generation**

| **Technique** | **Description** | **Examples** |
| --- | --- | --- |
| Equivalence Partitioning | Divides input data into valid and invalid partitions to test representative values. | Testing image files of supported formats (e.g., PNG, JPEG) and unsupported formats (e.g., PDF). |
| Boundary Value Analysis | Tests the edge values of input ranges to ensure correct handling of boundaries. | Testing maximum expected image dimensions (e.g., 1000x1000 pixels) and exceeding beyond (e.g., 2000x2000 pixels), and invalid dimensions (0x0 or negative values when trying to create a new image). |
| Error Guessing | Relies on the tester's experience to anticipate likely points of failure. | Testing invalid file paths or corrupted image files during loading. |
| State Transition Testing | Evaluates the system's behavior during state changes, such as undo/redo operations. | Testing multiple consecutive undo/redo actions. |
| Exploratory Testing | Involves manual testing without predefined test cases to uncover unexpected behavior. | Randomly applying multiple filters and layer operations to observe system responses. |

**6.4 Test Results and Assessments**

| **Test Case** | **Description** | **Expected Outcome** | **Actual Outcome** | **Status** |
| --- | --- | --- | --- | --- |
| Load Image | Load a PNG image from the local file system. | The image loads correctly and displays on the canvas. | Success | Passed |
| Load Unsupported File Format | Load a PDF file | Error message displayed. | Success | Passed |
| Edit Image | Draw a line on the canvas using the brush tool. | The line appears as expected on the canvas. | Success | Passed |
| Save Image | Save the edited image in PNG format. | The image is saved correctly to the specified location. | Not implemented | Failed |
| Manage Layers | Add, delete, and reorder layers. | Layer operations function correctly without errors. | Not implemented | Failed |
| Undo/Redo | Perform undo and redo operations. | Changes correctly revert and reapply. | Success | Passed |
| Apply Filter | Apply all implemented filters to the image. Test valid and invalid inputs for filters which take inputs. | The filter effect is visible on the image. Invalid inputs cause an error message and cancel the operation. | All filters apply successfully. | Passed |
| Large Image Performance | Load and edit an image of 1000x1000 pixels with 1 layer, and 500x500 pixels with 5 layers. | The system handles the image without lag or crashes. | Image handled successfully, but layers not implemented. Effectively, only single-layer images were tested. | Partially Passed |
| Invalid Brush Customization Inputs | Enter valid and invalid values (e.g., invalid color value, negative brush size). | The system prevents invalid input and displays an error message. Valid values are applied to the brush. | Color customization successful, brush size customization not implemented. | Partially Passed |

**6.5 Defects Reports**

| **Defect ID** | **Defect Description** | **Severity** | **Steps to Reproduce** | **Resolution** | **Status** |
| --- | --- | --- | --- | --- | --- |
| 001 | Draw tool doesn’t draw near edges when the mouse is dragged on or off of the image quickly. | Medium | Click on the image and hold to draw. Drag the mouse off of the image and back on at high speed. | Change line draw logic to allow lines to be drawn when one endpoint is off of the image. Check per pixel if the position is on the image. | Resolved |
| 002 | Minor lag when managing some images larger than 1000x1000 pixels. | Low | Load a large image and draw to it and apply filters. | Optimize image access handling. | Unresolved |
| 003 | For filters which take multiple inputs, dialog boxes after the first open underneath the main window. | Low | Use the “Add Color” filter and input a valid red value. | Determine if multiple inputs can be gotten within a single dialog window, or research the cause of the issue. | Unresolved |

**7 Conclusions**

**7.1 Outcomes of the Project**

The project successfully achieved most of its primary goals. The image editor includes many planned core features, such as image loading, drawing functionality, and filters. Due to the responsibilities of other classes, there was not sufficient time to implement some features, such as image saving, brush size customization, and layering. Minor limitations, such as performance with large images, do not detract from the overall functionality.

**7.2 Lessons Learned**

Key lessons include the importance of prioritizing essential features, effective teamwork, and regular testing to catch issues early. Balancing functionality and simplicity was critical to creating a user-friendly tool within the given constraints.

**7.3 Future Development**

Because this product is only a college project, the software is unlikely to be improved or maintained beyond the initial release.  
If further development is pursued, future enhancements could include:

Implementing features which were scrapped during initial development.

Support for more advanced filters and effects.

Broader file format compatibility.

Optimizations for handling larger images.

Additional tools, such as a selection tool or advanced layer options.