DEPARTMENT OF COMPUTER SCIENCE

Applying Flow-Based Visual Programming to Poster Graphic Design

Louis Decaudaveine

A dissertation submitted to the University of Bristol in accordance with the requirements of the degree of Bachelor of Science in the Faculty of Engineering.

Friday 5th May, 2023

Abstract

As an individual navigating the present-day digital landscape, I've noticed that users are constantly bombarded with vast amounts of information. I believe that structuring content in a way that optimises retention is crucial to effectively communicating a message to a targeted audience. I've found that studies have consistently shown that the fundamental elements of a poster, such as language and visual imagery, are more easily comprehended and retained by viewers.

However, while examining the software options available for poster design, I was surprised to discover that all the predominant applications offered either icon-driven or preset-driven interfaces. I saw a gap in the market for graphic design software that presents an innovative approach to the poster design process, through the implementation of a Visual Programming Environment(VPE) built for Poster Graphic Design.

My objective is to create a web-based application that integrates a bespoke Flow-Based Visual Programming Language for Poster Design. I want to present a unique approach to the poster design process that maintains a user interface that focuses on ease of use and presenting poster design concepts in an intuitive manner. Ultimately, I aim to conduct a user study to evaluate the impact of the VPE on the poster design process and assess the usability of the application.

Throughout the project I conducted the following:

- I researched in the fields of VPLs, FBP, Poster Graphic Design and relevant applications for the design of the Application
- I programmed three iterations of a web-application in Javascript that employs a custom VPL for Poster Design, with the aid of libraries such as React and p5.js and the framework Rete.jscs
- I conducted a user study to evaluate the application and the effect of a VPL on the user's Poster Design process.

Declaration

I declare that the work in this dissertation was carried out in accordance with the requirements of the University's Regulations and Code of Practice for Taught Programmes and that it has not been submitted for any other academic award. Except where indicated by specific reference in the text, this work is my own work. Work done in collaboration with, or with the assistance of others, is indicated as such. I have identified all material in this dissertation which is not my own work through appropriate referencing and acknowledgement. Where I have quoted or otherwise incorporated material which is the work of others, I have included the source in the references. Any views expressed in the dissertation, other than referenced material, are those of the author.

Louis Decaudaveine, Friday 5$^{\text{th}}$ May, 2023

Contents

List of Figures

Ethics Statement

This project fits within the scope of ethics application 0026, as reviewed by my supervisor, Simon Lock.

Notation and Acronyms

VPL	:	Visual Programming Language
FBP	:	Flow Based Programming
VPE	:	Visual Programming Environment
UI	:	User Interface
GUI	:	Graphical User Interface

Chapter 1

Introduction

Amidst the present-day digital landscape, where users are incessantly exposed with vast amounts of information, there is a pressing need to structure content in a way that optimises retention, with the ultimate goal of effectively communicating a message to a targeted audience. Multiple studies have consistently demonstrated that the fundamental elements of a poster, namely language and visual imagery, are comprehended and retained more effectively [10]. The central theme of this project revolves around the indispensable resources that facilitate the production of such media, specifically Graphic Design software.

Upon examining the software options available in the field, I was dismayed to discover that all the predominant applications solely offered either icon-driven or preset-driven interfaces. In the former, users employ design features abstracted through icons to design their products, while in the latter, pre-existing designs are modified to suit the user's needs. In light of the limited variation of Poster Design tools, I resolved to construct an application that presented an innovative approach to the graphic design process.

Upon examining software applications utilised in diverse domains of media creation, such as music production, 3D modeling, and video game engines, I noticed the incorporation of unconventional approaches to user interfaces, notably Flow-Based Visual Programming Languages, in contrast to the conventional icon-based interface. The rationale for adopting the VPL approach is supported by several advantages, including ease of comprehension and utilisation, a rise in productivity with increased familiarity over time [20], and the ability to exert fine-grained control over the design process.

However in the context of Graphic Design there does not exist an application that implements this type of user interface. Therefore to bring the advantages of VPLs to the world of Graphic Design and namely Poster Design, and provide a novel approach to the design process, the objective of this project is: To create a web-based application that integrates a Visual Programming Environment(VPE) designed specifically for Poster Graphic Design.

Initially, the primary aim of the project was to establish a VPL solely for the purpose of Poster Design. However, after undergoing several iterations of prototyping and considering the requirements of the intended end-users, the application was refined to achieve a distinct set of objectives, which can be enumerated as follows:

1. Cater to a broad spectrum of users, ranging from novice to expert graphic designers, as well as individuals with varying levels of proficiency in programming.

2. Presenting a unique approach to the Poster Design process through a Visual Programming Environment.

3. Maintain a user interface that focuses on ease of use and presenting Poster Design concepts in an intuitive manner.

4. Deploying the application in a suitable manner that guarantees accessibility to all intended users, to facilitate a user study aimed at analysing the impact of the VPL on the design process.

In order to attain the objectives of the application and generate the intended result, a number of steps will need to be undertaken. Firstly, it will be necessary to develop a custom VPL for Graphic Design, as there are currently no pre-existing ones available. This will entail obtaining an understanding of suitable coding paradigms to establish the foundation for the VPL, as well as investigating the features of VPLs and Poster Graphic Design, and conducting an analysis on the relevant tools accessible to the

general public. Moreover, it will be imperative to integrate a multitude of APIs into a coherent system to deliver the required accessible Visual Programming Environment to the end-users, facilitating the process of poster designing. Ultimately, upon completion of the application, a user study will be conducted to assess its performance and evaluate the effect of the VPL on the process of Poster Design.

The high-level objective of this project is to build and evaluate a Poster Graphic Design application that employs a bespoke Flow-Based Visual Programming Language, more specifically:

- Research and survey literature on VPLs, Flow-Based Programming(FBP), Graphic and Poster Design, and perform a review of relevant works to the project.
- Design and program iterations of the desired application starting at a proof of concept, to a minimum viable product, and finishing with a final product.
- Develop and conduct a user study of the final product, which assesses the effects of the VPL on the design process, the usability of the application and the impact of use time on the user's experience.

Chapter 2

Background

Given the nature of this project, before going into the details of the implementation, it is necessary to address certain topics first. Specifically, I need to contextualise Visual Programming Languages, its definitions, history and a relevant programming paradigm. Furthermore I will give a precise definition of Poster Graphic Design. Lastly I will provide a brief survey of the relevant works for this project.

2.1 Visual Programming Languages

2.1.1 Brief History of Visual Programming Languages

Sketchpad(1963), created by Andrew V. Sutherland is widely regarded as the first visual programming language. With the aid of a graphical user interface and a stylus, users could draw shapes and geometric figures. The visual programming aspect of it allowed users to create preset objects that could be reused and duplicated in other projects. Sutherland and Sketchpad later went on to win the 1988 Turing Award for his contribution to computer graphics[4].

The next advancement in VPLs came in 1968 with the release of GRaIL(Graphical Input Language) by Ellis et al.[5]. Users were able to draw flow-charts and specify their utility through the use of a stylus. The flow-charts are also represented as a written programming language that the user is able to manipulate with the stylus.

The concept of using flow-charts to represent a program was furthered by J Paul Morrison in the early 1970s with the creation and implementation of Flow-Based Programming in the IBM mainframes(AMPS) used by the Bank of Montreal[15]. In essence, FBP is a programming paradigm that is well-suited for visual representation, as will be explained later in this chapter. Morrison's contribution therefore remains more theoretical, where his ideas of FBP can be found in many contemporary systems, for instance NoFlo[15].

With the wide release of HyperCard in 1987 on all Apple Macintosh and Apple IIGS computers, VPLs had been brought to the masses on personal devices. HyperCard consisted of an embedded scripting language, user interface design tools and a multimedia authoring environment.

However as VPLs progressed and became more popular, the general consensus in 1995 was that VPLs "currently provide mechanisms to scale problem size but do not provide ways to optimize performance" Richard Frost[8]. This statement reflects the direction in which VPLs proceeded to be developed. Prototypes of Scratch were released in 2003 and 2004, and publicly released in 2007 to the world. Scratch is a block based visual programming language focused on teaching programming to children. Scratch being a web-application and being translated into over 70 different languages enabled it to reach millions of users since its inception.

Current day, advancements in hardware have made the previous claim redundant with the examples of many computer intensive fields using VPLs, for instance Unreal Engine, Touchdesigner, Blender and Maya all in the realm of 3D computer Graphics, utilise VPLs as a tool in their software.

2.1.2 Types of Visual Programming Languages

According to Marat Boshernitsan and Michael Downes [3], there are a few significant types of visual programming languages, here are the relevant types for this project:

Purely visual languages

These languages are characterised by how visual techniques are solely used throughout the programming process. A program is built through the manipulation of icons and other graphical components. If a debugger is offered it would also use the same visual environment. The program is then compiled and executed directly from its visual representation, and is not translated into a text-based intermediary language. The survey offered and, but dated back to 2004, therefore I have found a more relevant one: Unreal Engine Blueprint, when compiled, does not convert the visual elements into code, instead following a similar process to the C++ compiler, creates class properties and function lists, links the Classes and then compiles the functions and classes[6].

Hybrid text and visual systems

Here the visual elements are representations of underlying code. The system takes what has been made in the visual environment and turns it into code. The visual system acts as an extension of programming language. It can be argued that many IDEs are of this type, for example, the widely implemented feature of hyperlinks linked to variables and functions that enables the user to go to their definitions. However, visual programming languages of this nature may also include abstracted components that heavily depend on visual characteristics. For instance Unity's Visual Scripting, will take a piece of C# code and converts it into a visual node[13]. The node will display properties, inputs and outputs and notifies the developer of errors within the code through this visual environment.

Programming By Example

These systems consist of tools that enable the user to draw out their desired output, the system then takes the visual representation and converts it into code that replicates the users desired output. The aforementioned VPL type has seen significant advancements in recent years with developments of AI models. For example, Fronty, established in 2016, provides a VPL that intakes sketches of a desired website design and uses their AI to build the necessary HTML, CSS and JavaScript to replicate the design. More recently OpenAIs GPT 4, a large multi modal AI, has been applied to intake simplified hand drawn website designs and produce the necessary code[1]. Outside of the scope of AI, the 2010s saw the rise of NoCode systems, programs that enable the user to design technologies in a Program by Example manner. Significant applications of this nature are present in website development with the likes of Wix and SquareSpace, but are also applied to mobile App-development, Backend development for creating data structures and APIs[11].

2.2 Flow Based Programming

The visual programming language that I designed for this project is mainly inspired from the programming paradigm of Flow-based programming, for this reason it is essential to explain what FBP is.

2.2.1 Definitions

FBP being a programming paradigm focuses on the encapsulation of coding blocks called "black boxes" (most frequently being OOP objects), on the creation of modular and reusable code and on clarifying how data flows throughout a program. In FBP, a program is constructed by linking together "black boxes" to create a network, where information in the form of "IPs" is exchanged between these boxes. The nature of FBP lends itself to being visualised, where the encapsulated code is represented as nodes of a graph (as seen in Figure 2.1), the channels carrying the IPs between black boxes as lines or arrows that start from a port on the nodes and end on port of another node, and if the code encapsulated is an OOP object then chosen properties of it can be visualised within the node.

Black Box

In classic FBP, a black box is a process which can be run as a coroutine, asynchronously to all other black boxes. Therefore making a Flow-Based program inherently a concurrent program. The main purpose of a black box is to exchange and process data, they may or may not have input ports and output ports where IPs are received and sent respectively through channels. Black boxes can be interconnected endlessly to form the desired functionality. Furthermore,a key feature of FBP is the capability of compounding a

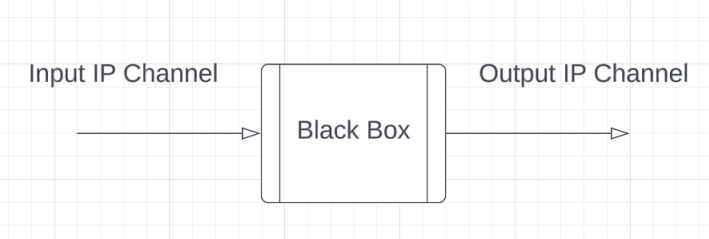

Figure 2.1: Diagram of a "Black Box"

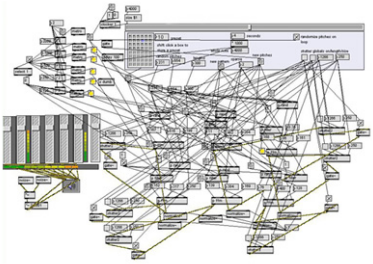

Figure 2.2: Cluttered Flow-Based program made in Max MSP

Flow-based network into a black box. This enables a developer to build on previous work and follows the concept of reusability present in FBP.

Information Packets(IPs)

J Paul Morrison, the author of seminal book on FBP "Flow-based Programming: A New Approach to Application Development", describes IPs as an atomic concept that cannot be decomposed into more fundamental objects. It represents the information that is processed through the handles of a flow-based system. They are predominantly data but not always, for instance they may be signals that are passed from one black box to another. IPs can be shared and exchanged through the channels that connect the black boxes, to become a "stream", which are produced and consumed asynchronously.

2.2.2 Benefits and Drawbacks of FBP

Firstly, the concurrent nature of FBP, abstracts away the need for the user of a Flow-Based language to consider concurrency and coroutines, therefore leading to parallel programs that execute code more efficiently and saves time on data processing. However it is important to state that parallel programming does necessarily equate to optimised programs. Furthermore, multi-threading is not available in all development environments. A classical FBP language developed in such an environment would therefore not be possible.

As stated previously, Flow-Based programs are restricted to reusable components (Black Boxes). The modular approach forces developers into producing programs that inherently have smaller code bases. This can be seen as a benefit and as a drawback. A benefit, as it reduces time spent navigating the code bases since they are smaller. A drawback, as implementing unintended features of a Flow-Based language can be impossible or require excess boilerplate code that nullifies the benefits of reusable code.

By utilising the theory of External Cognition, specifically external representation[16], and with the established assumption that the visualisation of a Flow-Based program externalises data flow through the channels connecting the "Black Boxes", it can be implied that FBP benefits from greater clarity of data flow throughout a program. However in practice, Flow-Based programs with extremely large networks can end up cluttering the visual programming environment, leading to abstruse programs, as seen in Figure 2.2. However Figure 2.3, shows how such a program of the same size can be manipulated to support the previous claim. Therefore, FBP does not inherently benefit from clarity of data flow, but has the capability to.

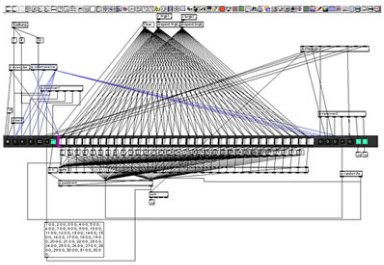

Figure 2.3: Organised Flow-Based program made in Max MSP

FBP-inspired systems

Classical FBP was first conceived in the 1970s and the programming environment has substantially changed since then. Therefore recent Flow-Based languages have also adapted to the current state of programming, adapting the FBP model to include new technologies. A relevant shift in FBP was the creation of Flow-Based languages built off JavaScript with its single threaded nature and dynamic type checking, for instance Node-Red or NoFlo. This has inspired FBP systems in which the components no longer run in parallel, additionally to accommodate for a single threaded implementation, the components are controlled by their upstream connections rather than being autonomous[14]. However inspired systems vary in many ways, the main takeaway is that classical FBP remains the foundation of Flow-Based systems but the systems are not restricted by its principals.

2.3 Poster and Graphic Design

Having established the essential information regarding VPLs, it is imperative to define the concept of Graphic Design and its specialized area, namely Poster Design.

2.3.1 Graphic Design

Graphic Design has been defined as

> "a craft where professionals create visual content to communicate messages. By applying visual hierarchy and page layout techniques, designers use typography and pictures to meet users' specific needs and focus on the logic of displaying elements in interactive designs, to optimize the user experience." [7]

Graphic Design remains a interdisciplinary branch of design and of the fine arts. The first recorded academic course in the field dates back the 1917, where Frederick H. Meyer, an instructor at the California School of Arts and Crafts, taught a course called "Graphic Design and Lettering"[17]. The discipline is a collection of sub-disciplines, significant ones being: typography, poster design, book and periodical design, interaction design, illustration, exhibition design, branding and corporate identity[19].The profession predates computers and therefore was originally performed through physical means. However, since then many digital tools have been developed for the purpose of Graphic Design. Over the past 5 year, Adobe, as one of the major software companies responsible for developing these tools, has experienced continuous growth[2]. Therefore showing an increasing amount of designers are using digital tools.

Graphic Design continues to be a wide-ranging discipline, with a considerable portion of its practice devoted to the realm of fine art.Thus, given the scope of this project, I have decided to ignore this aspect and focus on the Human Computer Interaction between the software and the designer. Additionally to be able implement a holistic tool given the time and resource constraints the project will focus on Poster Design.

2.3.2 Poster Design

Poster Design is a sub-discipline of Graphic Design and focuses on the design of posters. Following the Cambridge Dictionary a poster is

> "a large printed picture, photograph, or notice that you stick or pin to a wall or board, usually for decoration or to advertise something"

To expand on this definition, traditional posters generally include textual and graphical elements, however they may also consists solely of either. Poster Design is not a recent craft but the advent of digital tools and digital posters in the field has changed its definition. What once was restricted by the binds of printing, is no longer. Posters can now be multimedia, consisting of audio, text, video, digital art, photography and animation.

Posters find a range of applications in various contexts. Presented below are some instances of their utility:

- Educational Posters: In academia, posters are a form of promotion for an academics research, whilst also providing an abridged holistic explanation of it.
- Event Posters: One of the most common application of a poster is for events. Used as a form of promotion, it enables events ranging from protests to concerts to be advertised and share the necessary information.
- Film, TV Series, Book and Comic Posters: Similar to Event Posters these types of posters are used to generate interest for the media, however they focus on marketing over information.
- Product Poster: The primary aim of a product poster is to promote and advertise a product to potential customers. Such posters can be in the form of a conventional print-out or a multimedia format.

Whilst the aforementioned applications to poster designing mostly remain accessible to only professional Graphic Designers. Recently new accessible tools have been developed to facilitate individuals with limited experience to partake in graphic designing and inherently poster designing, for instance the free web-application Figma initially released in 2016 or Canva founded in 2013. These tools that present themselves as welcoming to beginners or non designers yet also capable of providing professional results, raises the question: How can a graphic design software appeal to professionals and beginners?

2.4 Relevant Works

During the project's developmental phase, I often consulted pertinent programs and software to take inspiration for features that proved to be valuable for both novice and expert users, additionally to decide on the essential design tools that should be implemented. Given that my project is centered on Flow-Based VPLs and Poster Design software, and since no existing tool has integrated these two aspects simultaneously, when investigating relevant works I address each aspect separately.

2.4.1 Poster Design Tools

Figma

Figma is a free collaborative web-application focused on interface design and white-boarding, although within the interface design aspect, it contains a significant amount of features enabling Poster Design.

Figma provides the user with a range of typography utilities in a similar icon layout native to text editors, whilst remaining versatile and precise. For instance you may change the spacing between each individual letter. Furthermore, Figma has a range of Vector Based tools, meaning that zooming or increasing the size Vector Based elements does not change its definition, an essential tool for poster designing since posters are shared in many different sizes. One of the main selling points of Figma is its collaborative nature, enabling a group of users, in real time, to edit a design.

I chose to include Figma because of its accessibility to novice users alongside expert users. The layout of the application is simple, when starting a new design there are few icons presented in the work environment, as the user adds more elements to the design, an increasing amount of features are offered, enabling the user to explore what is available without getting overwhelmed when starting out. Likewise, this approach of building the environment as the design becomes more complex accommodates expert users by enabling them to reach more complex designs with ease.

Present in Figma and in the other mentioned Poster Design Tools, are features that follow Ben Shneiderman's HCI principal of Direct Manipulation [18]. This principle proposes that users can control

software behaviour by manipulating objects on a screen to make it more intuitive. Examples of manipulations consist of: Drag and dropping objects to perform an action, for instance dragging an object off the screen to delete it; pinching objects to perform an action, for instance pinching a rectangle on a canvas to increase its size. All of these manipulations enable rapid incremental feedback to the user and is essential in Poster Design for modifying the layout of a poster.

Canva

Canva is a free web-based application and software that offers a graphic design platform, which concentrates on on template oriented designing.

Similar to Figma, Canva has many different applications, such as : a PDF and MP4 converter and an Image Enhancer. However it also advertises itself as being capable of Poster Design. The approach to design that Canva employs is driven through templates. Instead of starting on an empty canvas Canva recommends a series of presets that the user can apply and edit. For instance if the user is designing a film poster, the user would pick an appropriate template then edit the placeholder text and change the poster background image. Canva enables the user to nest presets within themselves, for instance you can nest an image framing preset within a business card design preset. Additionally, the application provides the user with a library of vector art, stock images and videos, that can be employed into the designs through Direct Manipulation.

Canva is included in this sections for the opposite reasons to Figma. In my opinion, the application exclusively accommodates for novice users. Canva lets a novice user achieve a final product rapidly through the merits of a preset's efficiency. However the lack of precision tools on the application leads to users not being able to fine tune their designs and limits the flexibility desired by expert users. The key insight gained from Canva remains the efficiency of presets.

Adobe InDesign

InDesign is the industry-leading page layout designing software application. As an Adobe product, it establishes links between all InDesign projects and other applications of the Adobe Creative Suite, thus making it one of the most powerful Poster Designing products available.

InDesign is characterized by an extensive range of features that would be challenging to comprehensively enumerate in this section. The application's layout is similar to that of Figma, however instead of revealing more features as a user's design becomes more complex, the work highlights a majority of them through the GUI. The work environment is split into four sections : A column of icons representing actions the user can perform; a nested drop-down header with more specific actions, a sub-header with a mix of icons and input fields and a column of dynamic custom interfaces that the user can manipulate and organise.

I chose to include InDesign, for its custom interfaces column and more specifically the layers component present in it. Due to the complexity of the InDesign application, it exceeds the scale of my project. However the unique layer component that it possesses provides a tangible representation of the design process, and incorporates various features. The order of the layers translates to how you would layer a collage. If a layer A is under a layer B , and A and B overlap, B will be displayed in full as A is partially displayed, the overlapping segment of A will be hidden under B. InDesign lets the user: group components withing layers, merge layers into one, reorganise the ordering of the layers, hide the layers within the design and use layers as clipping masks.

2.4.2 Visual Programming Languages

TouchDesigner

TouchDesigner is node-based Visual Programming Language designed to build real-time interactive multimedia content, falling under the "Creative Coding" umbrella term. TouchDesigner can be used for : rendering and compositing 2D or 3D scenes generated from the VPL; building GUI applications that can be employed for multimedia installations and video mapping.

The TouchDesigner work environment comprises three distinct sections: a minimalist header section utilized to establish the rest of the environment, a body section housing the VPL editor and rendered scenes, and a footer section dedicated to managing performance and temporal progression within a scene. For the context of this project I will focus on the body of the work environment. The VPL is similar to that of a Flow-Based VPL, the user builds their program through placing, editing and connecting nodes to form a network. The nodes are holders of Operators, which come in 6 types: Object Components,

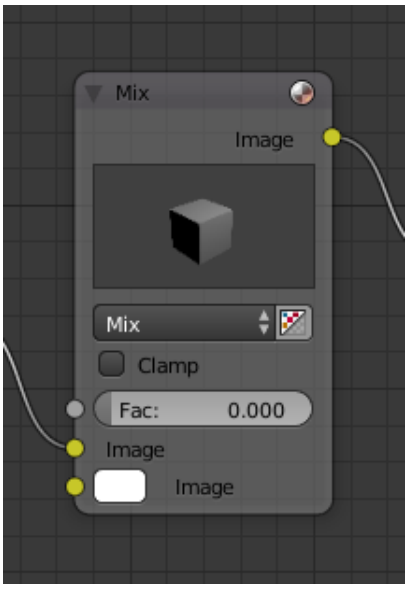

Figure 2.4: Example Node in Blender Node Editor

Texture Operators, Channel Operators, Surface Operators, Data Operators, Material Operators. Each node may have input ports and output ports that enable them to be connected to each other. The user can manipulate the properties of a node through its icon menu.

There are three features which I found essential for my project that are present in TouchDesigner's VPL:

- The VPL editor itself and the manner in which a user can navigate it. The editor canvas is endless, therefore enabling the user to organise the position of the nodes spaciously and clearly to avoid cluttered networks mentioned previously. Additionally the zooming feature of the editor facilitates abstraction when programming, when zoomed out all the user can perceive is name of the node and to what it is connected to, when zoomed in the user is able to identify precise properties of the node, for instance a preview of the geometry if the node is an object component.

- The Operator menu in which a user chooses nodes to add to the editor. The menu can be brought up through right clicking at a point on the editor and that is where the new node will be placed. The menu is divided into sub-menus, making it easier to navigate. Having a node menu present on the VPL editor I found streamlined the programming process.

- Visualising data in the nodes. When processing data through a VPL network, being able to see how the node affects the data helps in constructing and debugging a network with the users desired output, it is similar to being able to access the state of a specific object in Object-Oriented Programming.

Blender: Node Editor

Blender is a free 3D Computer Graphics application software, providing a software tool set for 3D modelling, UV mapping, texturing, physics simulations, rendering, editing and many more features outside of the scope of this project. However a section of these tools are manipulated in the Blender Node Editor, a node-based work environment.

The Node Editor has multiple use cases, for instance Compositing and Material and Texture generation. The networks built in the editor shares many characteristics with that of FBP inspired systems. Each node within a node tree is controlled by their upstream connections whilst waiting on their downstream connections for data. The flow of data traverses channels and is sent and received through typed sockets, similar to FBP IPs.

The reason I chose to include the Node Editor was for its presentation of nodes within a Flow-Based system. The primary disadvantage of TouchDesigner nodes is the limited transparency of data-flow within a network, particularly when examining the data transfer and processing at the level of individual nodes. In the case of Blender the systemic node layout - in which the inputs are on the bottom left

of the node, the outputs on the top right, the title at the top and the properties and previews in the body of node - follows the principal of Consistency in HCI. Therefore increasing the users efficiency [12]. Furthermore, the nodes in Blender explicitly show the sockets of the nodes with colour coded circles that represent the type of the socket, as seen in Figure 2.4. This clearly informs the programmer which sockets are compatible with each other, and removes the ambiguity of node connections that is present in TouchDesigner.

Chapter 3

Project Execution

During the evaluation of existing Graphic Design systems and investigation of the concepts and uses of flow-based Visual Programming Languages, my project underwent two iterations before reaching its final design. The goal of the project was to produce a flow-based VPL for poster designing under the form of a web-application. Furthermore, my objective was to create a program that catered to both novice and expert graphic designers by implementing a novel approach to poster design, specifically a poster designing VPL.

I have organised this chapter into multiple sections, the first being an overview of technologies used to execute the project, followed by three sections focused on each iteration's design and implementation.

Note all the figures with captions ending with a (*) are made using the final product and the corresponding VPL program can be found in the Appendix

3.1 Technologies Used

Programming Languages

The project being a web-application led me to choose to program the project in HTML,CSS and JavaScript. I picked these three languages for the vast amount of documentation, libraries an support that is available online. HTML was used for creating the foundational structure and content of project, CSS to produce the layout and design of the program and JavaScript to handle the VPL and Graphic Rendering along with the rest of the projects logic.

React

React is a front-end JavaScript library used to build user interfaces. React applications are built through the use of custom components that developers program. In the case of this project I used React to combine the graphic rendering and the VPL editor into a cohesive UI. My selection of React was based on its flexibility, the convenience it provides when integrating different libraries to the application and its extensive documentation.

p5.js and react-p5

p5.js is a JavaScript library, a graphics API that was used throughout the project to render the intended poster designs of the user. It provides a set of tools for creating and manipulating graphics, animations, and sound. react-p5 is a JavaScript package for integrating p5.js into a React application. I chose p5.js as the graphics API of this project, since it is open source and free to use and because of the wealth of documentation and support.

Rete.js

Rete.js is a JavaScript framework that enables the construction of flow-based VPLs that can be run on web-applications. The framework provides developers with : a Visual Programming Environment, a node construction API and serializable features that enable the conversion of VPL programs to JSON and vice versa. I used this framework in the development of the projects VPL.

Figure 3.1: VPL node design for Prototype I (*)

react-beautiful-dnd

react-beautiful-dnd is a JavaScript library compatible with the React library, serving as a tool to make drag-and-drop lists within React components. This library was used in the final iteration of the project to help develop the layers features.

npm

npm, Node Package Manager, is a package manager for JavaScript. It allows developers to install, manage and share reusable packages of code, as well as provides developers with a local development environment to deploy websites locally. I used npm in the project for the aforementioned reasons.

GitHub

GitHub is a cloud based git repository. I used it throughout the project for version control and as storage for the project's code base.

Netlify

Netlify is a cloud based deployment, and server-less back-end services for web applications and dynamic websites. I used Netlify at the end of my project to deploy the web-application for evaluation an general use.

3.2 Prototype I: Proof of Concept

The initial prototype was developed at the outset of the project, during a period of uncertainty regarding the desired functionality of the VPL. Additionally it is the prototype that received the least amount of development time.

3.2.1 Design

The idea for the first prototype was to build a VPL that acted as a visual aid to FBP using the p5.js library. By creating and interconnecting a set of custom flow-based JavaScript objects within the p5.js setup function, visual components representing these objects would be rendered along with the graphic. The user would then be able to fine tune the graphic through the Visual Programming Environment.

Prototype I was never intended to be used within the final version therefore the design of this version shares more similarities with a programming library then a web-application. It was designed as an extension of p5.js providing the aforementioned capabilities under the form of objects that can be instantiated to fit the users requirement.

In this first iteration, the VPL design remained simple. As seen in Figure 3.1 - the figure being made using the final product - the VPL node is a rectangle that consists of a title in which the user can manipulate the nodes position through clicking and dragging it to the desired destination and of editable properties of the underlying flow-based object, for fine tuning the graphic. Note the lack of sockets and

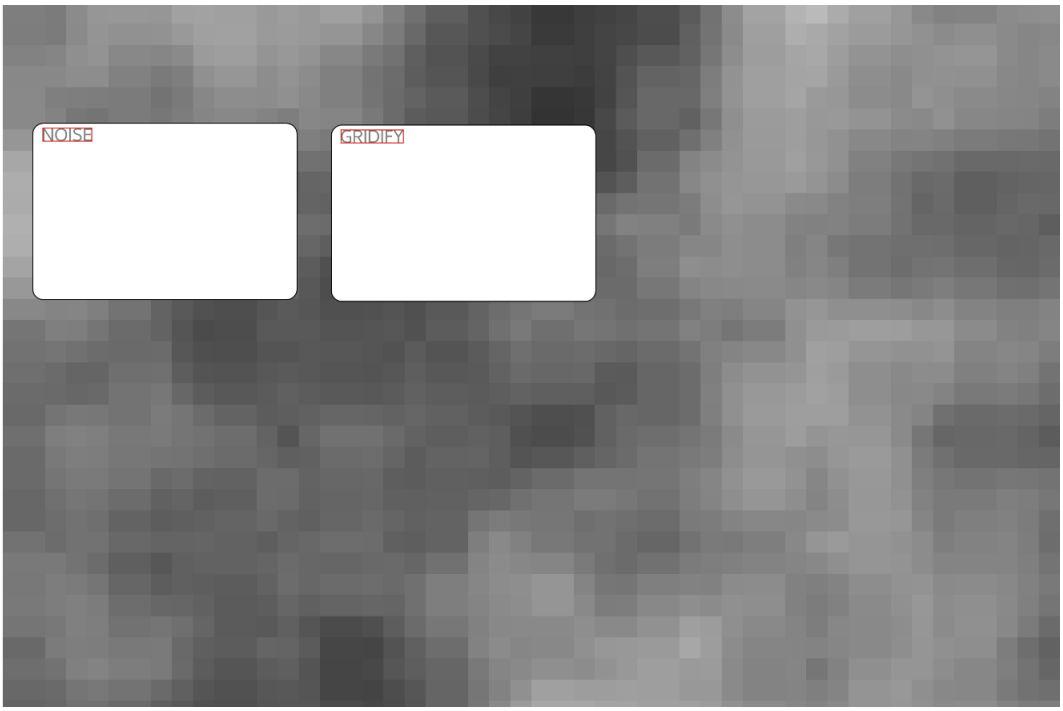

Figure 3.2: Instance of a design using Prototype I

channel visual representation, this was purposefully designed in such a way since this prototype only was designed to have two different VPL components, *Noise* and *Gridify*. Therefore, there is only one possible connection that can be made and the connection may remain ambiguous visually.

The intended use cycle of this prototype consisted of the following steps:

1. The user starts a new design by creating a new p5.js project using the prototype's flow-based objects to build the graphic.
2. Once the user was satisfied with his code they runs the program, which would render the design and the visual programming environment.
3. The user then edits the properties of their flow-based objects within the VPL to fine tune the design
4. If the user is satisfied with their design they then save it, if not they go back to the first step with the updated state of the objects.

Regarding this project, the first prototype ended up serving as a proof-of-concept for testing the feasibility of utilising FBP as a solution for poster design, as well as to gain knowledge and experience in website development.

3.2.2 Implementation

The implementation of the VPL was split into two parts, the visual programming environment and the VPL node components.

I first programmed the parent node component object *PCB*. This component was inherited by all the other VPL components. It handled all the fundamental node functionalities: position on the visual programming environment, movement functionalities of the node, the title and size of the node. Additionally I programmed two more components: *NoiseBlock* and *GridifyBlock*. *NoiseBlock* enabled the construction of *Noise* nodes which outputted either a Perlin *Noise* two dimensional matrix or a one dimensional value of Perlin *Noise* to its connected nodes. A *Noise* node would be instantiated within the p5.js setup function and then rendered using further p5.js functions. *GridifyBlock* enabled the construction of a *Gridify* node which had both an input and a graphical output. The input consisted of a reference to an object, in which outputs could copy their data to, in this case the data was a two dimensional matrix of JavaScript numbers. *Gridify* nodes processed the values of the matrix and outputs an array of p5.js functions to

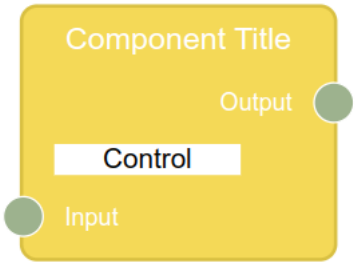

Figure 3.3: Example Component Code Built in Rete.js

render the graphic. *Gridify* nodes would be instantiated the same way as a *Gridify* node, with the added aspect that the graphical output of the node needs to be called in the p5.js render function.

For the visual programming environment, I overlapped the environment over the generated graphic design. The main feature of the environment was the capability to move the nodes around, in the aim of helping maintain VPL organisation through Direct Manipulation. To achieve this I implemented event handlers for mouse clicking on the website to check if the mouse was positioned over any node, if it was then the aforementioned move functions would be called from the parent node components.

Finally implementing the graphics rendering consisted of using the rendering functions of p5.js that were each called from the output array of the *Gridify*. In the context of this restricted implementation the graphics rendered were a series of rectangles that represented Perlin Noise on a two dimensional plane. The end product of this prototype can be seen in Figure 3.2.

Figure 3.2 highlights the absence of editable properties pertaining to the flow-based objects represented in the VPL nodes. This discrepancy between the design and implementation arose from the movement of development from Prototype I to Prototype II, leading to the incomplete nature of Prototype I's implementation.

I made the decision to progress from Prototype I and construct a fresh code-base utilising novel technologies that I had investigated, namely, npm, React, and Rete.js. Additionally, I revised the concept of my envisioned VPL, choosing instead to incorporate the VPL into a web-application.

3.3 Prototype II: Minimum Viable Product

Drawing on the lessons learned in the previous prototype, I started the development process anew, with the goal of making a minimum viable product, in which the prototype would be capable of basic poster designing through a VPL. The second prototype was designed with a dual focus on a web-based application comprising two essential components: firstly, the development of a basic visual programming language using Rete.js for poster design, and secondly, the integration of a graphic renderer using p5.js that could interpret basic programs written in the VPL into the graphical elements of the poster. Furthermore, I expected that this prototype was not going to be my last, therefore I designed the prototype to be run locally and to not host it on a server, since I was to be the primary user and did not necessitate the tool to be accessible to others.

3.3.1 Design

3.3.2 VPL

Since the VPL design is constructed on the Rete.js framework for this prototype, I will initially provide a brief exposition of the Rete.js architecture prior to expounding upon the VPL design. Rete.js provides the developer with a series of tools to build flow-based VPLs.

The first tool being the visual programming environment(VPE), the VPE is an infinitely sized canvas in which a user can zoom in and out endlessly and can navigate the canvas by left-clicking and dragging along the empty canvas, similar to that of the editor of TouchDesigner.

Additionally, Rete.js provides JavaScript functions and classes to build custom *Components* that can be instantiated and visualised as *Nodes* within the VPE. Figure 3.3 is an example *Component Node* built using the Rete.js framework, the design of the nodes is split into four sections: the input and output

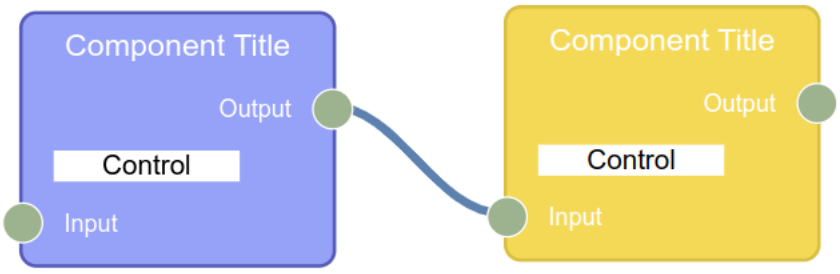

Figure 3.4: Example of connected nodes in Rete.js

sockets, the title and controls, as shown in the figure. The roles of each are the same as that of the previous prototype except for the addition of the controls. Controls in Rete.js can be designed in many forms, they are HTML elements that have access to the nodes underlying object properties, and can be input fields to modify these properties. The nodes can be manipulated within the VPE through the same drag-and-drop approach native to TouchDesigner and Blender. *Nodes* can be connected to each other by drag-and-dropping your mouse from one input to one output, and is visualised through Bezier-curves, as seen in Figure 3.4. The visual connection of nodes is an essential feature for flow-based VPLs as it visualises the flow of data.

The *Context Menu* within Rete.js is another factor in my decision to utilise this framework. The context menu consists of two situational drop-down menus that can be toggled on either a node or on the canvas of the VPE. If toggled on the node, a menu appears at the mouse position that offers to duplicate or delete the node. If toggled on the canvas a menu appears at the mouse position with a list of components, all which cause the creation of component nodes. Both menus can be modified through the framework. The combination of these menus, the VPE, the nodes, the controls and the visual connections, provides the sufficient features needed to build a program from a VPL. All that is needed now, is a design for the content of the VPL. Before expanding on the content needed for the VPL, its important to note that Rete.js has two more essential tools - the engine and the serializer - that I will elaborate on in the implementation segment of this prototype.

For the second prototype, I chose to design the components of the VPL as concepts that could be represented graphically, for instance a rectangle or a colour component. The rationale for making this decision was to enable a prototype development process that prioritizes the workflow from VPL to graphic renderer and vice versa, thereby allowing for a holistic approach to the construction of the project.

I incorporated a consistent root node - the *Sketch* node - into the design of the VPL, ensuring that all programs constructed within the VPL always have a uniform starting point. The *Sketch* node, was designed so that the user could manipulate the essential metadata of the graphic renderer. Furthermore, it serving as a uniform starting point for all programs was intended to create consistency in how programs were read in the editor. Where a user building a poster could descend its corresponding program from the root node through its input's connections to figure out the steps taken to reach the current state of the poster. Finally, the *Sketch* node has one input socket that accepts data that can be visualised in the renderer.

The other components developed for this prototype will be explored in the implementation of the prototype.

The Renderer and Application Workflow

Unlike the first prototype, the VPL programs are not built in the same environment as the p5.js code that makes up the rendered poster. Therefore for this prototype, I needed to design a system that enabled the parsing and conversion of data between the Rete.js VPL environment and the p5.js graphics renderer. The solution I designed was to have an interpreter for the VPL, in which the VPL program is converted into a series of JavaScript objects that are called within the p5.js renderer, as seen in Figure 3.5. The *Application* component depicted in the figure serves as the central hub of the project, responsible for the web-application layout design, data storage, and execution of essential processes to bridge the gap between the VPL and the rendered output.

Furthermore, since I want to incorporate Direct Manipulation within the rendered Poster, such that

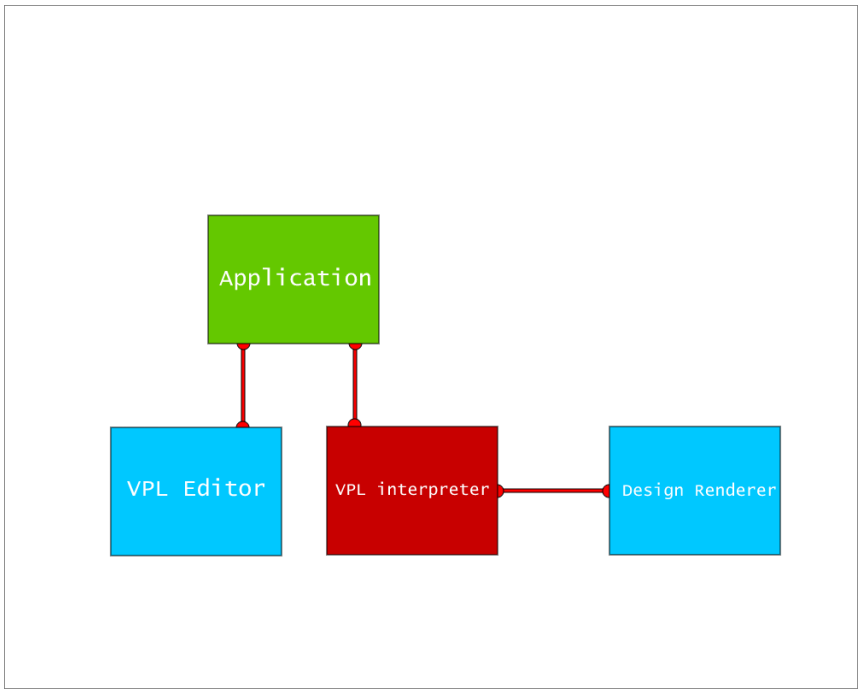

Figure 3.5: Diagram of Prototype II workflow (*)

the user can fine tune the layout of their poster designs, the interpreter needs to be capable of also converting from the JavaScript objects into its corresponding VPL program.

The *Design Renderer* in Figure 3.5 primarily is designed to handle the p5.js graphics API. Such that it processes the aforementioned JavaScript objects, updates them every render cycle and renders the necessary graphics. Alongside rendering the designs, *Design Renderer* handles the direct manipulation of visual components within the designs through the use of bounding boxes. The concept of bounding boxes within this project is such that each visual component is encapsulated within an invisible rectangle. The bounding box of a visual component is the area in which the mouse has to be positioned to move the visual component through a drag-and-dropping action with the mouse.

The user use cycle of the second prototype has distinctly changed from that of the first prototype, therefore here is the updated use cycle of the web-application:

1. The user starts off by initiating a local server and running the web-application on it. The user then visit the local server on their web-browser of choice, where they are presented with the VPL editor containing one node, the *Sketch* node.

2. The user is capable of initiating new nodes and interconnecting them through the use of the *Context Menu*. When satisfied with the state of the VPL program, the user can render the design through saving the program (In the implementation this is under the form of an HTML button).

3. The user is offered the option of manipulating the the visual components through their bounding boxes, which updates the state of its corresponding VPL nodes. If the user is satisfied with the design they can save it, if not then they can loop back to step 2.

3.3.3 Implementation

In comparison to the other iterations, a relatively substantial portion of the project's developmental timeline was allocated towards Prototype II. This was due to the fact that I had edited, expanded and detailed my projects design alongside having to dedicate time to comprehend the programming environment consisting of a new framework and new libraries.

The implementation commenced through the initialisation of a React project, in which I created the necessary React components to wrap the Rete.js VPL environment and the p5.js Graphics API environment. To adhere to the original design depicted in Figure 3.5 and to exercise greater control over the mutability of each environment, I opted to enforce isolation between both environments - in the form

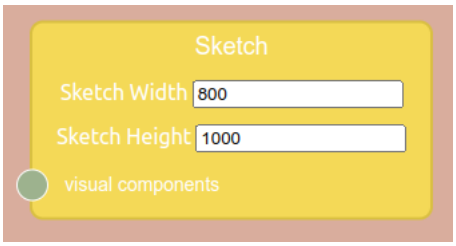

Figure 3.6: VPL Sketch Node from Prototype II

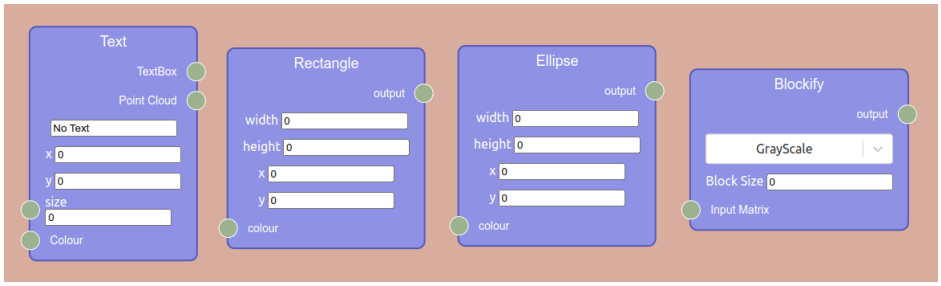

Figure 3.7: VPL Visual Nodes from Prototype II

of the React components - using React's robust *State* and *Props* features to store and exchange data between them.

VPL Controls

Subsequently, I started developing the controls for the VPL nodes. The controls serve as a way of portraying and editing the underlying data of the node objects. The most widely used control was the "Number" control, consisting of a custom HTML *input* tag, which could either be editable or strictly informative, the numbers could be restricted to integers or floats. Furthermore I implemented a "Text" control which like the "Number" control, could be editable or strictly informative, alongside being built off a custom HTML *input* tag. Taking inspiration from the preset philosophy seen in Canva, I decided to implement a drop-down control, which had the functionality of changing a property of its node's underlying object to a selection of alternatives present in the drop-down. It was built from a React drop-down component. The last control I implemented in this prototype was a colour preview, which was built off a custom HTML *div* tag, and consisted of a coloured square that matches its given hexadecimal colour value.

All of the Controls were built as JavaScript sub-classes of the Rete.js framework *Control* class. These sub-classes were composed of a logic segment that was called within the Rete.js engine and a JavaScript XML element that served as the control element in the nodes they were a part of.

VPL Components

With the implementation of these controls, I was able to build the VPL components. I started off by making the *Sketch* component, comprised of two "Number" controls and an input socket, representing the width and height of the rendered graphic. The component node is instantiated on every VPE, as seen in Figure 3.6. Creating new components through the Rete.js is achieved through the creation of sub-classes of its *Component* class. As mentioned previously, I focused on developing components representing visual concepts. Therefore, following the same development process as the development of the *Sketch* component I built *Rectangle, Ellipse, Text* and *Blockify* components, as seen in Figure 3.7. In the first three components, the user can choose the position, dimensions and colour, and in the *Text* component there is additionally a text input field. The *Blockify* component serves as the counterpart to the *Gridify* component in the initial prototype. Nevertheless, it now includes a "Number" control to choose the size of the blocks and a drop-down of stylistic presets for the visual output. Additionally, its input socket exclusively accepts data in the form of a two-dimensional array of floating-point values.

On top of the visual components, I programmed two more none-visual components: the *Colour* and *Noise* Components, as seen in Figure 3.8. The *Colour* component consists of three integer input fields

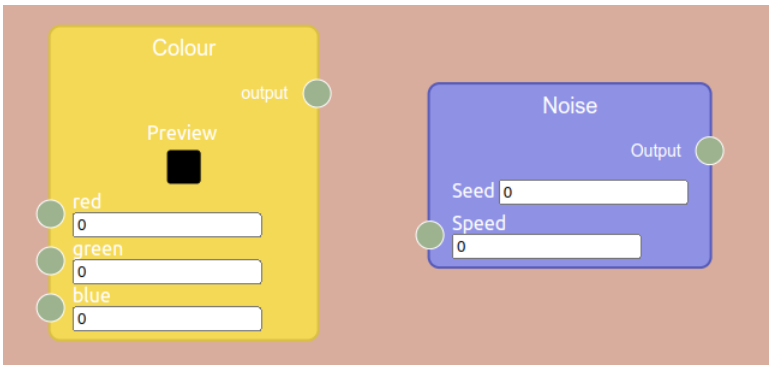

Figure 3.8: VPL Non-Visual Nodes from Prototype II

representing the RGB value of the desired colour, and of a colour preview. The *Noise* component is built with two number input fields intended to represent the seed and speed of traversal of a three dimensional simplex-noise matrix. The execution of simplex-noise operations does not take place within the VPL component itself. Instead, these operations are carried out in the corresponding *Design Renderer* JavaScript object. The underlying reason for this lies with the compatibility of Rete.js engine and of the web-application.

The Rete.js engine enables the transfer and processing of data from connected nodes within the VPE. Every time a "Process" event is triggered from a node, a series of downstream updates are first triggered (all nodes connected through inputs), followed by upstream updates (all the connected nodes from the output of the node). This cascading approach to updating the nodes of the program is efficient in most scenarios, since it accommodates to a wide variety of network structures. However in the context of this prototype, it would have to process and transfer data of objects in the *Design Renderer*, and that would require the Rete.js framework and p5.js Graphics API to work in tandem, which I found was not possible. Consequently, I restrict the utilisation of the Rete.js engine to only manage the essential data required to generate previews and values displayed in the controls.

VPL Sockets

Through the process of building the VPL components I needed to build the typed sockets that enable the restricted connection of nodes through the inputs and outputs of the nodes. Here is a table of all the sockets that were built :

List of Sockets		
Socket	**Status**	**Description**
numSocket	In use	Accepts any type of number
colourSocket	In use	Accepts any type of colour object
floatArray2DSocket	In use	Accepts any two dimensional array of floats
colourArray2DSocket	Discontinued	Accepts two dimensional array of colour objects
textSocket	In use	Accepts string values
coordArray	Discontinued	Accepts array of coordinate objects
testSocket	Discontinued	Used in development to help build the interpreter
ListSocket	Discontinued	Accepts any socket type, equivalent to a generic
VisualSocket	In use	Accepts any type of object that can be visually rendered

It should be noted that nearly half of the table comprises of discontinued sockets. This can be largely attributed to the time constraint that prevented the implementation of features that would necessitate their use. The only exception is the testSocket, which was utilised during the *Interpreter* construction process.

Interpreter

Differing from the intended design shown in Figure 3.5, the *Interpreter* and *Design Renderer* have been joined into one React component during the implementation of Prototype II. The *Interpreter* component in this prototype remained partially implemented, meaning that it could only convert VPL programs with a maximum graph depth of two layers, but for the available VPL components of this version, the VPL programs could only reach a depth of two layers.

To use the Interpreter I had to develop a series of JavaScript objects that could be instantiated and function with the p5.js graphics API. To accomplish this, I adopted the approach of creating a distinct object for each VPL component, thereby facilitating a one-to-one conversion of nodes into the new objects. In this version of the project, the new objects were of two types: Visual Objects and Non-Visual Objects. The Non-Visual Objects, consisted of an ID, a data variable in which the corresponding control data of the VPL program could be copied to, and the necessary generated data for the particular object, alongside an update function that could be called in the rendering of the design. The Visual Objects shared the same structure as the Non-Visual Objects with the addition of the aforementioned bounding boxes, and the necessary p5.js rendering functions to be called when used in the rendering section of the code.

The be able to convert the VPL programs, I utilised the Rete.js serializer. The serializer performs the conversion of the programs into a JSON format that could be parsed as an Object in JavaScript. With the serialized object I traversed through all of the inputs of the *Sketch* node, looking at what type of component they were derived from instantiating the correct p5.js compatible object and placing the object reference into a layers and an objects lists. The *Interpreter* then checked if the *Sketch* inputs, had any inputs of their own, repeating the same process to build the corresponding Non-Visual Objects, and adding the references into the objects list.

To handle any changes from the *Design Renderer* to the VPL program, the *Interpreter* matches the serialized VPL node ID with its p5.js compatible object's counterpart and replaces the corresponding serialised data with the objects data. With this modified serialized VPL program, Rete.js provides the functionality of building a VPL program from the serialized object, therefore the *Interpreter* calls the necessary *Application* handler functions to build the VPL from serialized.

Design Renderer

The *Design Renderer* serves three discrete purposes. On the first render of the Design, it loads in all the required assets, for instance the font, and builds the graphics canvas with the given size of the *Sketch* node. Once the first render has finished, the *Design Renderer* enters a render cycle offered by the p5.js functions, in which it iterates through the *Interpreter*'s objects list to call the update functions of the objects, followed by the layers list to call the objects' render functions. Finally I implemented the necessary functions to provide Direct Manipulation of the visual components, through the use of the graphics API's input event handling, in which I update the Visual Objects position and bounding boxes when the appropriate conditions are met - that being the mouse position is over the bounding box and the mouse is pressed down and dragged to a new position.

The implementation of Prototype II was complete and fulfilled all the necessary features set out to be implemented within this minimum viable product.The prototype consisted of a basic VPL for poster designing, capable of simple typography and graphic generation, alongside a design renderer capable of formatting the poster through the user's visual manipulation.

3.4 Final Product

While Prototype II was able to fulfill the initial design requirements that I had established, it was found to be deficient in various aspects that are deemed essential for a comprehensive Poster Designing application. In particular, there was a need to enhance the VPL to accommodate the multi-modal aspects of poster creation, as well as to incorporate the concept of layers that is commonly found in other relevant works in the field and essential for formatting. The goals of the final phase of the project remained open ended. Although there were some pending features that needed to be integrated into the application's framework, the majority of the foundational groundwork had already been accomplished. The main focus was to maximize the incorporation of additional functionalities within the given time constraints.

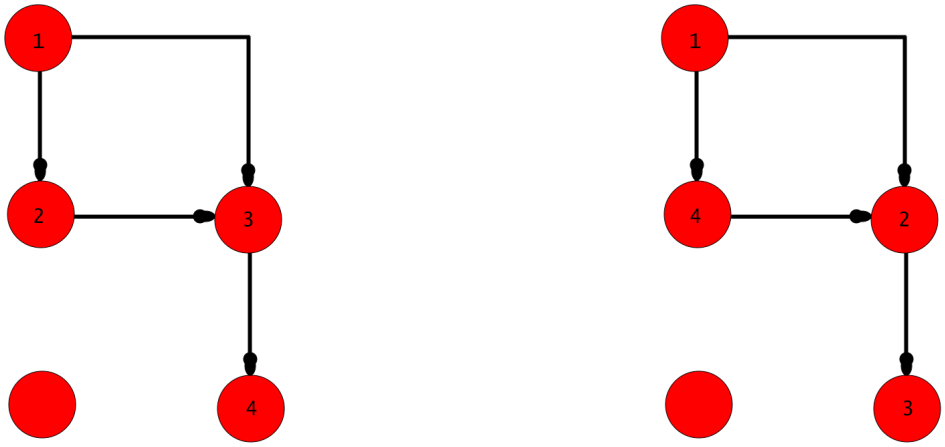

Figure 3.9: Depth First Search Possible Node Orders in a Directed Graph (*)

3.4.1 Design

The design of the Final Product, is built off the previous prototype, and the framework that was developed for it, therefore, in this version, my emphasis was on formulating features that brought me closer to achieving the overall goals of this project. Presented below is a concise list of discrete objectives that steered the development of the Final Product:

- The creation of a Poster Design tool featuring a streamlined GUI aimed to mitigate user disorientation and enhance ease of use.
- The development of a Poster Design tool designed to accommodate both novice and proficient users in the domains of programming and graphic design.
- Building a Poster Design tool that lends itself to the iterative and incremental development [9] of posters.
- Novel approach to Poster Design that expert graphic designers have yet to experience.
- Building a multi-modal dynamic Poster Design tool, which lets users add their media to the designs and animate the visual components.
- Building an intuitive VPL in which utilities can be inferred visually.

Interpreter

Certain elements of the previous design were revised for this final version. As stated previously the interpreter was incomplete as it only functioned on VPL programs with a graph depth of two levels. The latest version of the application has been augmented with additional features that enable the VPL programs to attain graph depth levels without any imposed limitation, therefore the interpreter needs to be updated to accommodate for the updated VPL.

For the new design of the interpreter I considered that all VPL programs fall under the form of directed graphs, where the edges of the graph are the connections between the node and the direction of the edges are from input socket to output socket. The interpreter is designed to follow a Depth First Search inspired algorithm to instantiate the appropriate objects starting from the *Sketch* node. I chose this approach as only the nodes which are directly and indirectly connected to the root node are visited as seen in Figure 3.10 and instantiated into the correct objects, avoiding any unnecessary node not used in the poster design. Furthermore the DFS algorithm can be used in graphs with any level of depth and therefore will accommodate any sized VPL program.

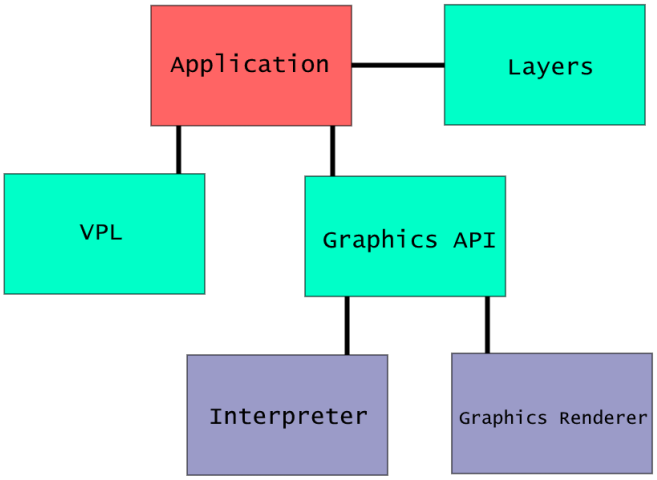

Figure 3.10: Final Product Application Workflow (*)

VPL

In the context of the VPL, I conceived a collection of mathematical components with the aim of giving users the ability to create dynamic posters. These components enable the manipulation of visual elements when connected, resulting in a dynamic visual output. I designed an Addition and a Multiplication components, which are also capable of subtraction and division, a sine component which is capable of giving a sine or cosine value at a particular point and a Step component, intended as a value that gets incremented every render cycle. The process of incorporating these functions into the VPL's design, necessitated modifications to the behavior of the visual components. Specifically, the introduction of input ports to the controls of components, enabling the representation of values as either static input values or the dynamic mathematical expressions provided through the input sockets.

To add to the multi-modal features desired in a Poster Designing application, I conceived a new file explorer control for the VPL, in which the user is capable of adding their own media to the posters. This control was applied to the design of the Image component. The Image component enables the user to add their own images to the poster.

Upon examining TouchDesigner in the context of this project, I encountered difficulties in identifying the interconnections between its various components. However, upon revisiting Prototype II, I discerned that the previous iteration's input and output sockets were indeterminate. In light of this, I resolved to design a color-coding system that would unambiguously demarcate each type of socket and facilitate the connection between compatible ports. This colour-coding was intended to be applied to the sockets on the nodes of the programs.

Layers

As articulated in the aforementioned objectives, my intention was to adopt an iterative and incremental approach towards the development of the posters. This entails ensuring that the application use cycle is straightforward and succinct, such that the user is not confronted with sweeping modifications to the poster. Instead, users can make slight modifications to the VPL, fine-tune the visual components in the renderer, and repeat the process. However, in Prototype II, achieving the desired overlap of visual components is not feasible since all the visual components are fixed in their layering. Consequently, to achieve the intended outcome, all the visual components have to be simultaneously linked to the *Sketch* node in the correct order, which runs counter to the original goal. Therefore I decided to design a *Layers* React component which enabled the reordering of layering order within Visual Components.

The incorporation of the *Layers* component resulted in a modification to the rendering process of the Visual Components as seen in Figure 3.10. Specifically, rather than arbitrarily iterating through the list of layers in the *Graphics Renderer*, the *Application* component now stores the layers in a designated order, and a duplicate is dispatched to the *Graphics Render* whenever a mutation occurs.

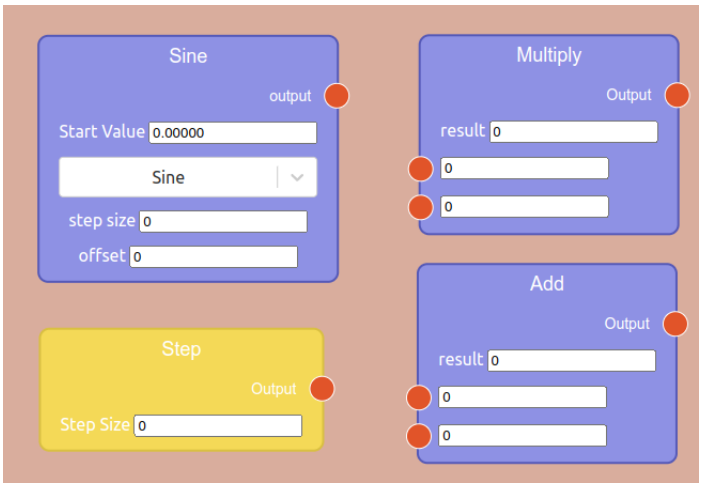

Figure 3.11: VPL Mathematics Nodes

Application Deployment

As this was the final iteration of the product, and I intended the project to be widely accessible to the general public, the Final Design was designed to be deployed on the internet through the aforementioned cloud based deployment service Netlify. This was also a deliberate strategy to broaden the pool of candidates for evaluation, due to the ease with which it could be configured for use on personal devices, through a hyperlink.

In addition, I aspire to incorporate a tutorial video in the application, which will serve the dual purpose of introducing new users to the software and functioning as a point of reference for troubleshooting.

3.4.2 Implementation

VPL

For the last version of my project, I wanted the user's to be able to animate the visual components through mathematical expressions built from VPL components. I therefore implemented the previously mentioned four components as seen in Figure 3.11. Nonetheless, during the implementation phase, I encountered a challenge which required the operations to be executed within the VPE. This was necessary to ensure that users did not connect the nodes blindly without knowledge of the resulting expressions. As a result, I employed the Rete.js engine to construct the essential logic for generating previews of the expressions at each stage of an expression's building process. Additionally, this reinforces a process of incremental development in the VPL programs of the user.

Once having implemented the set of components I needed to incorporate them with the rest of the VPL component suite. Hence, I transformed several controls of the previously implemented components into conditional input ports that acted as controls when no connection was established, and accepted the input value when a connection was established. Due to my inability to overcome the issue of conflicting animated positions and direct manipulation of visual components, in the time constraints of the project, I was unable to complete this integration.

Implementing the *Image* component in Rete.js and in p5.js, consisted first of building a *File Explorer* control that could store file data within the application in a format compatible with the graphics API. The solution to this was to store the files as *Blob* objects. Once the *File Explorer* control was implemented, the Image component and object were built in parallel to make sure that the file formats were compatible. Ultimately, the image files are uploaded to the *Image* components within the VPE and transferred to the Graphics API, where only the critical image metadata is stored in the VPE to populate the preview controls of the nodes, as seen in Figure 3.12. I opted to transfer the files to minimise the size of the serialised VPL programs, which are generated multiple times during the poster design procedure, thus ensuring the efficient operation of the application.

To implement the colour coded sockets, I built a series of custom CSS styling classes focused on each typed socket HTML element. Therefore for each instantiated VPL node, the colour of the sockets were filled with distinct colours representing their types.

Figure 3.12: Instance of an Image Node

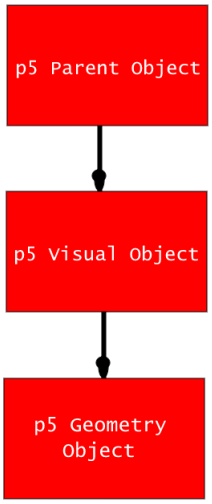

Figure 3.13: Object Hierarchy of p5.js Parent Objects (*)

Layers

In the current project, the implementation of the *Layers* React component involved utilising the react-beautiful-dnd library to create a modifiable ordered list. This list comprises HTML components that represent the currently rendered Visual components. The library adds the functionality of turning these HTML components into Drag-and-Droppable objects, enabling the user to intuitively reorder the lists through Direct Manipulation. The underlying ordered data is stored within the *Application* component seen in Figure 3.8, and a copy is sent to the *Graphic Renderer* every time a layer is moved or added in the *Layers* component.

Interpreter and Design Renderer

To enhance the efficiency of future feature implementations and accommodate the growing number of components in the VPL that require p5.js compatible objects, I decided to refactor said objects. In the refactor I chose to adapt the concept of class hierarchy within Object-Oriented Programming with a series of parent JavaScript objects, in which the p5.js compatible objects inherit from. Figure 3.13 shows the three objects that were made. *p5 Parent Object* provides the necessary functionalities of all p5.js compatible objects. The *p5 Visual Object* provides all the necessary functionalities of Visual Components, including the initialisation of bounding boxes, and the *p5 Geometry Object* focuses on implementing the necessary functionalities of geometry Visual Objects, such as the Rectangle Object and Ellipse Object, both inherit from. Additionally, some of the objects have updates which can take a substantial amount of time. To improve the performance of the *Design Render* I modified all the update functions to be asynchronous, therefore making sure that the renderer does not remain stuck updating on one of these

objects, every render cycle.

Within the *Interpreter* I rebuilt the VPL serialised program conversion to accommodate for multi-level program graphs. I accomplished this by integrating recursive functions that adhere to the Directed Depth-First-Search algorithm, which begins at *Sketch* node.

Deploying the Application

Due to time constraints, I concluded the development of the project and opted to deploy the web-application on a third-party platform to make it widely accessible to other users. To accomplish this task, I utilized the Netlify service. I generated a build folder through npm and subsequently uploaded it to the cloud-based deployment tool provided by Netlify. Hence, my project is now accessible for evaluation via the following link: https://vocal-baklava-85f016.netlify.app/.

Chapter 4

Critical Evaluation

In order to assess the efficacy of the project, I opted to conduct a user study on the web-application. The user study constituted a means of quantifying both the application's usability and the practical implications of its innovative approach towards Poster Design. The study required participants to use the application to produce a poster. Subsequently, participants were asked to fill out a questionnaire that gauged their experience while utilizing the application.

4.1 Participants

The user study was originally intended to encompass individuals possessing diverse levels of proficiency in both Graphic Design and programming. The rationale behind this decision was the application's intended purpose of being accessible to both novice and expert Graphic Designers. Moreover, given that the tool relies on programming concepts, it was imperative to measure the experience of individuals with varying levels of familiarity with programming.

Given the uncomplicated setup requirements of the project's application environment, I was capable of reaching a wide range of participants for the study. In this regard, I distributed emails to both students and professionals with varying levels of proficiency in Graphic Design and programming, with a focus on maximizing outreach to a broad audience rather than a limited one with guaranteed participation.

The outcome of the participation of the user study was deficient in terms of the diversity of skills exhibited by the participating individuals. Within the questionnaire, participants were provided with the opportunity to express their level of proficiency in both Graphic Design and programming. The responses from all nine participants are as follows in Figures 4.1 and 4.2. In both domains, more than half of the participants self-evaluated their proficiency level as being at the novice stage. Notably, only one participant rated their Graphic Design skills as professional, and a lone individual expressed near employable levels of programming skills. Therefore, it can be observed that the user study will predominantly focus on novice programmers and graphic designers.

Figure 4.1: User Study Histogram

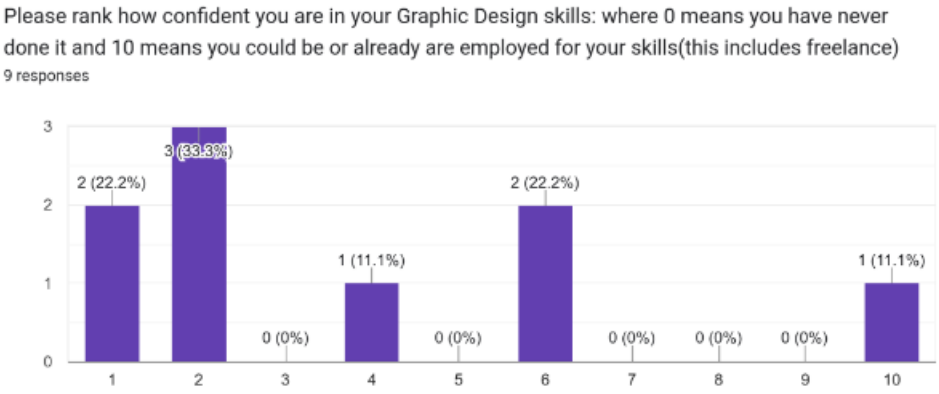

Figure 4.2: User Study Histogram

4.2 Procedure

Each participant was sent an email with a link to the user study instructions and questionnaire. The participants could perform the study in any environment as long as they had access to a computer, a stable internet connection and reserving a minimum of ten minutes to partake in the study. Once opened, they were first asked if they consented to partake in an anonymous study, then they were directed to the web-application through a hyperlink, and offered the opportunity to watch an embedded tutorial video briefly explaining the purpose and functionalities offered by the application. Watching the tutorial video was optional, however every participant ended up watching it. The core constituent of the user study was then presented to the participants.

To make the users get the intended experience that was designed for in the application, the participants were tasked with designing a poster of their choice, with the possibility of restarting their designs over. Following the definition of a poster, provided in the Chapter 2, the posters were not restricted to a specific field (e.g., a film poster) or form of media. The duration of the participants' engagement with the program was at their discretion, under the stipulation that they maintain a record of the time spent. I opted not to constrain the duration of application usage in order to compare the survey responses among individuals with varying usage periods. Upon reaching a satisfactory poster design or reaching a point of saturation with the application, the participant was requested to complete a survey.

The survey that the participants had to fill out, consisted of 22 questions comprising three different answer forms: multiple choice answers, short text answers and long form text answers. Moreover, specific questions were designed to elicit responses only if specific conditions were met in previously answered questions. The survey questions could be categorized into three distinct aims: acquiring comprehension of the pertinent background of the participants, gauging their perception regarding the application's usability, and examining the impact of a VPL on their graphic design process. During the survey creation and data collection process I utilised Google Forms.

Ultimately, I chose not to request the posters produced by the participants during the user study in order to maintain the anonymity of the study. Additionally, the analysis of the resulting posters falls within the domain of Graphic Design and is beyond the purview of this project.

4.3 Results

This segment will be partitioned into three distinct sections, each concentrating on different constituents of the user study findings. Initially, the responses pertaining to the usability of the application will be deliberated, followed by those concerning the effect of the VPL on the graphic design process of the participants. Finally, an assessment of the influence or absence thereof, of the duration spent utilising the application on the participants answers.

4.3.1 Usability of the Application

Initially the participants were asked to judge the overall usability of the application ranging from one to five where one represented "Very Poor" and five "Excellent" . The findings presented in Figure 4.3 indicate that 77.8% of the participants assigned a rating of four, indicating their perception of the

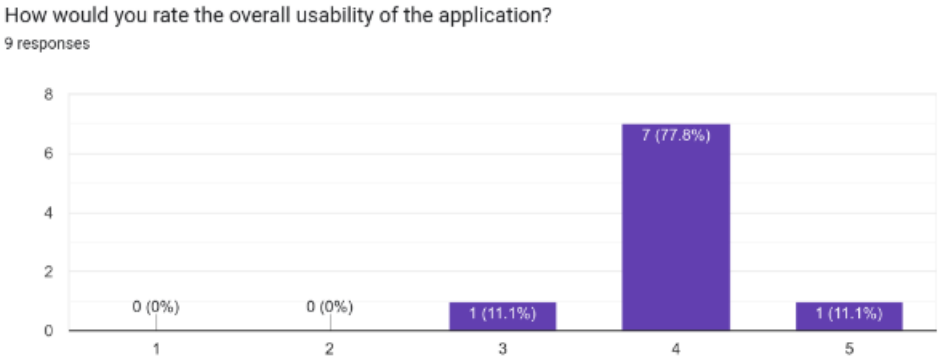

Figure 4.3: Histogram on Overall Usability

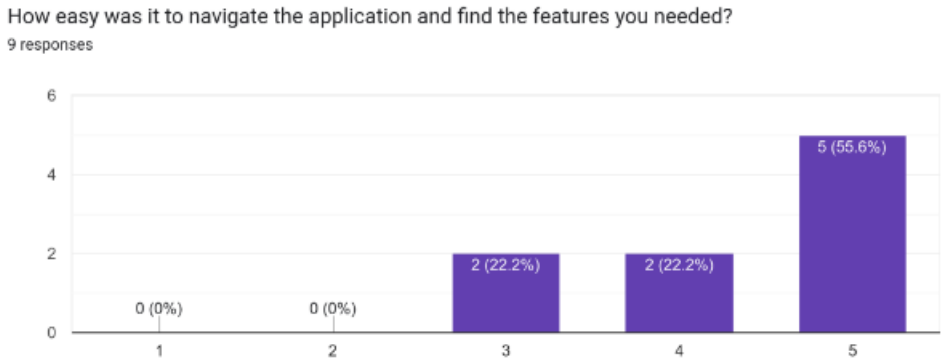

Figure 4.4: Histogram on Application Navigation

application as being more than adequate in relation to usability. Nonetheless, given the comprehensiveness of the notion of overall usability - ease of use - this rating does not furnish significant detailed insights. Therefore, within the survey, there was a series of questions focusing on more precise elements of the usability. Navigational ease among features is a fundamental aspect of application usability. When participants were queried about the ease of navigation within the application, with responses ranging from one to five, where one denoted "very difficult" and five represented "very easy", 55% of responses were five, and the remaining were above three as seen in Figure 4.4. Additionally, when asked about the overwhelming nature of the user interface, 77.8% replied with "No, never" while the remainder responded with "Occasionally." These outcomes affirm that the tool's ease of use is influenced by the presentation of its features and therefore the overall usability of the application.

Another factor that contributes to the usability of this project is whether users encountered any technical issues while utilising the application. Technical issues, being beyond the user's control, can hinder task completion, thereby intrinsically impacting the usability of the application. For this question the participants were offered three possible answers: "Yes, frequently", "Occasionally", "No, Never", when asked if they faced any technical issues with the application. Figure 4.5 depicts the outcomes, with 55.6% of respondents indicating "No, never," and 44.4% responding with "Occasionally." These results suggest that technical issues could have hindered the application's full functionality for nearly half of the participants, negatively impacting the overall usability metrics.

The responses from participants regarding the usability of the application, as evident from the series of questions posed, can be deemed successful, given that the majority of answers indicate a favorable opinion towards the ease of use of the application, attributable to the uncomplicated nature of the user interface and feature navigation, as well as the absence of overwhelming complexity in the user interface and design process.

4.3.2 Impact of the VPL on the Graphic Design Process

An essential component for this project was to bring a novel approach to the poster designing process, through the implementation of a VPL. Therefore some of the questions asked in the questionnaire con-

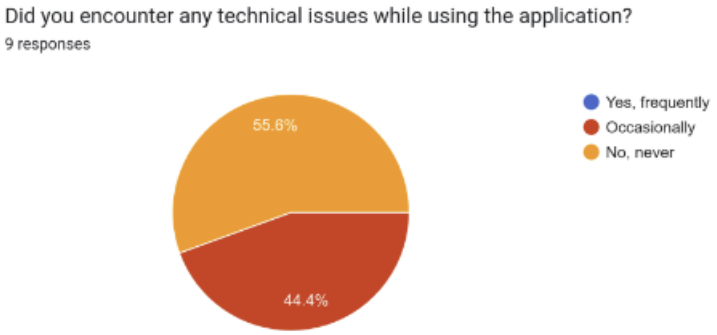

Figure 4.5: Pie Chart about the Technical Issues relating to the Application

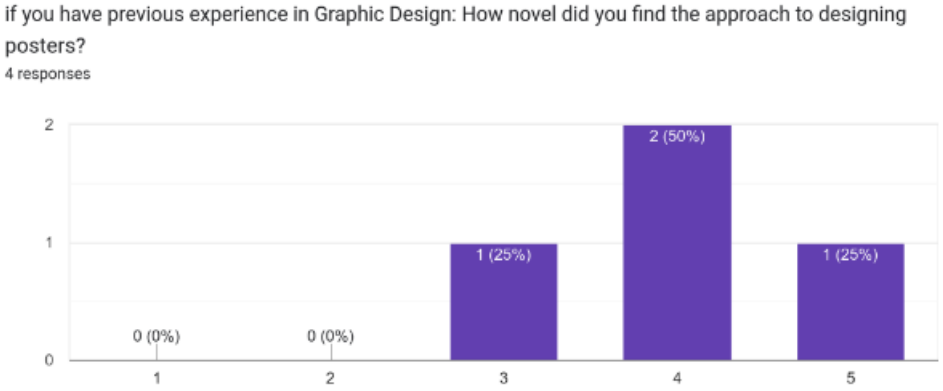

Figure 4.6: Histogram relating to the novelty of the project

sisted of looking to what extent the VPL affected the poster designing process of the participants. Within the user study the participants with previous Graphic Design experience were asked to rate how novel they found this approach ranging from one, representing "Common", to five, representing "Novel", Figure 4.6 illustrates the results. Out of the nine participants four of them answered the question. The results range from a score of three to five, with 50% of answers being a four. This suggests that individuals with a background in Graphic Design perceive the VPL approach as innovative.

Given that this approach is predicated on FBP, I sought to investigate whether users with prior coding expertise could apply their programming skills to the process of designing posters. Regrettably, only two participants were able to respond to this query in this study. As a result, when queried about the degree to which they were able to apply their programming proficiency to this application, one participant responded with a rating of three, while the other responded with a rating of four, on a scale ranging from one denoting "Not transferable" to five representing "Extremely transferable". Based on the small sample size, it can be tentatively inferred that the VPL approach facilitates the transfer of skills for individuals with prior coding experience.

Another factor that I examined to measure the impact of the VPL approach was its relative ease of learning compared to other Graphic Design tools. As seen in Figure 4.7 80% of the five participants stated it was just as difficult and 20% stated that it was easier. The VPL approach can be inferred to neither significantly facilitate nor impede the learning process compared to other Graphic Design tools.

During the survey all the participants were asked if they would choose to use this tool over other Graphic Design software available followed by the same question but instead of this application, an application that shared the VPL approach with a wider range of features available. For the former question 55.6% of participants said they would rather pick my application, and for the latter 100% said that they would pick an application that shares the same VPL approach. Notably, among those who rated their Graphic Design abilities as above five and selected my application in the former question, they constituted 25% of the subgroup. Thus, it can be deduced that the application is well-suited for beginners in the field of graphic design, but lacks the necessary features to appeal to intermediate and advanced users. Nevertheless, the VPL approach appears to be a promising method for graphic design that can be used by individuals of all skill levels.

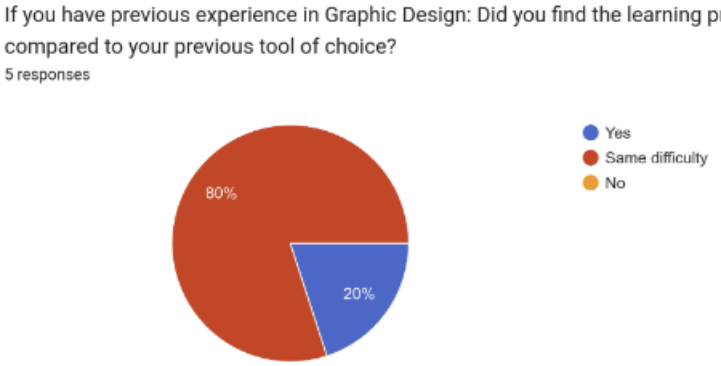

Figure 4.7: Pie Chart depicting the difficulty of the learning process

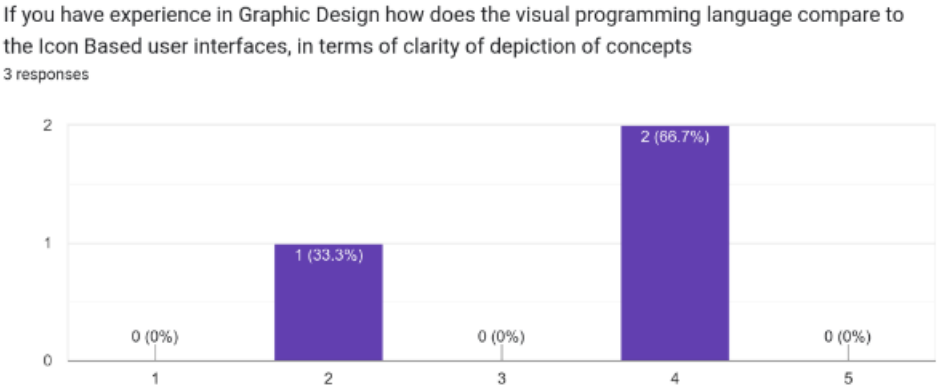

Figure 4.8: Histogram depicting the GUI preference of participants

Additionally, I aimed to compare the VPL approach with an icon-based layout to determine which approach represents the features' concepts more effectively. In this instance, I chose to implement a five-point Likert scale as a means of gauging the degree to which participants' attitudes were aligned with either available option. Here one represented a preference for the VPL approach and five for the icon-based user interface. As illustrated in Figure 4.8, it can be observed that two respondents opted for a rating of four, signifying a stronger inclination towards the Icon-Based approach, while one participant indicated a rating of two, indicating a divergent perspective. It can be deducted that those with previous Graphic Design experience find the icon-based user interface a more effective way of deducing a features utility.

Ultimately, two additional questions pertaining to the VPL were posed. The initial query sought to determine the level of intuitiveness or perplexity experienced by participants while constructing a poster using the VPL, while the subsequent inquiry aimed to assess the extent to which the participants evaluated the quality of the posters produced via the application. For the first question I used a Likert scale ranging from one, representing "Confusing", to five, representing "Easy to understand". The outcomes presented in Figure 4.9 demonstrate that there was no definitive inference to be made, as participants' evaluations of the VPL's intuitiveness were evenly dispersed across a range of ratings from two to five. For the second question, a Likert scale was used again this time ranging from one, being "Very poor", to ten, being "Excellent". The responses obtained for this query were situated within the range of five to eight, with a majority of 88.9% of the participants registering ratings within this interval, as depicted in Figure 4.10. Hence, it can be inferred that, according to the opinions of the participants, the VPL facilitates the generation of posters that are of a higher quality than their average poster design standards.

The feedback provided in relation to the impact of the VPL on the process of poster designing reveals a varied response with regards to its intended function and effectiveness. Certain aspects of the VPL surpassed the anticipated objectives, as evidenced by the unanimous preference expressed by all participants for a more refined and comprehensive application that incorporates the VPL approach, in comparison to other graphic design software options. On the other hand, the VPL did not fully achieve certain objectives, particularly with regards to clarifying the representation of concepts related to the

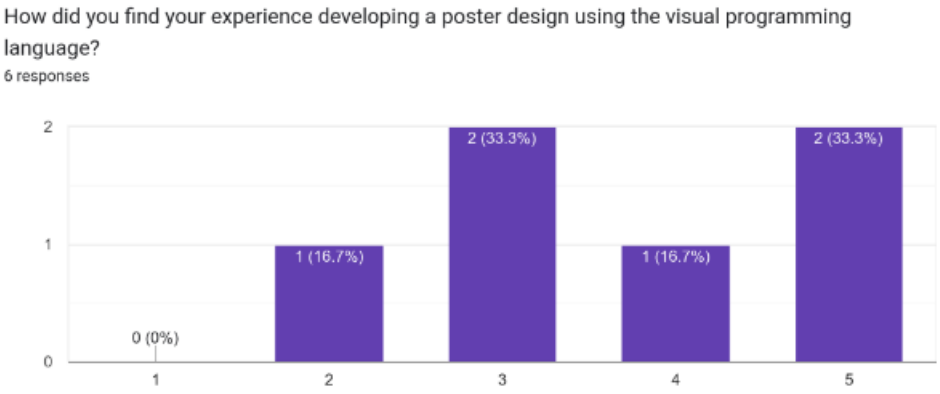

Figure 4.9: Histogram depicting the intuitiveness of the VPL

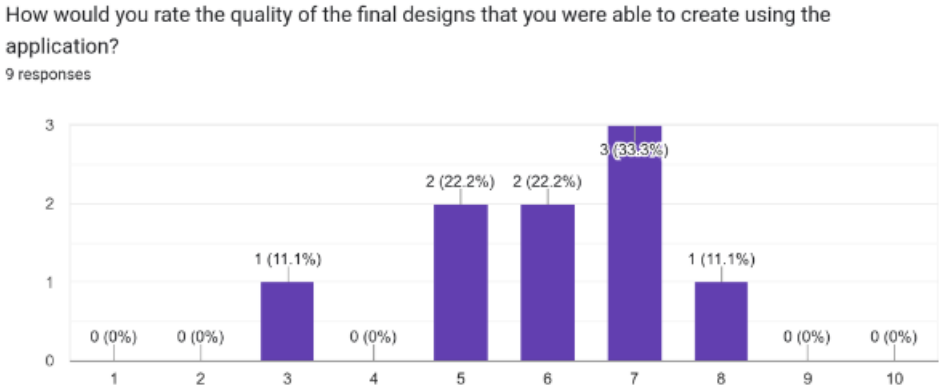

Figure 4.10: Histogram depicting the intuitiveness of the VPL

features of graphic design software.

4.3.3 Impact of Time of Usage on the Results

As previously stated, there was no time limitation imposed on the participants for the user study. The application use time of the participants, depicted in Figure 4.11, hovered around 20 minutes, ranging from 10 to 45 minutes. This variance was deliberate, with the objective of determining whether the duration of usage had a discernible effect on the users' experience, as evidenced by the survey results obtained from varying durations.

After reviewing the results in relation to the duration of application use, there were only two fields in which the results were impacted. The first being the frequency of technical issues and how often the participants found the user interface overwhelming.

In Figure 4.12 the answers relating to the frequency of technical issues are depicted in a Bubble Chart, with the size of the bubble referring to the amount of participants that shared the same answer and use time. The analysis indicates that in cases where participants reported experiencing occasional technical problems, half of them were attributable to the two users who spent the most time utilising the application, among the four cases under consideration. This outcome is unsurprising, given that the probability of encountering technical issues increases with the duration of software usage. However, it prompts an inquiry into whether this set of answers is biased, and if extending the participants' duration of interaction with the application would bias the results toward a higher frequency of responses in the "Yes, frequently" and "Occasionally" categories, as opposed to "No, never" and "Occasionally.".

Regarding the outcomes for the inquiry about how overwhelming the user interface is, Figure 4.13 presents a Bubble Chart that illustrates the responses with respect to the duration of usage, utilising colour to convey the frequency of participants that shared the same answers and duration. For this question only the two users that spent the most time on the application found the user interface occasionally overwhelming compared to the rest which never did. Presumably, this is a consequence of the complexity of the VPL programs produced, which renders the VPE overwhelming. Comparable occurrences have

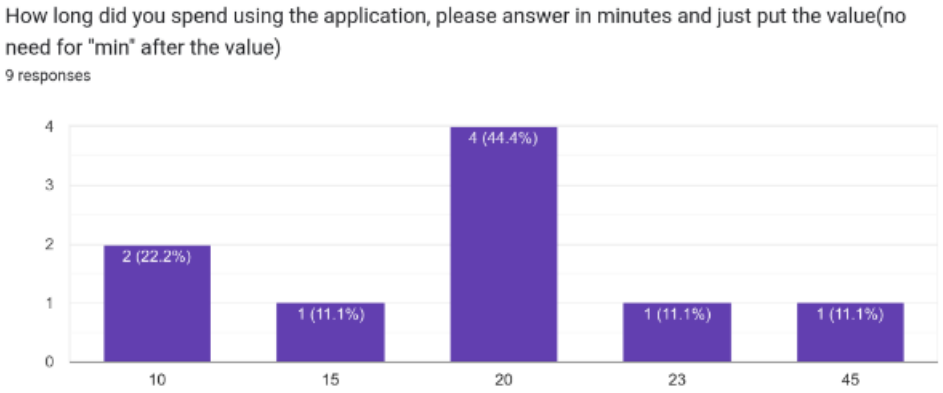

Figure 4.11: Histogram depicting time spent on application

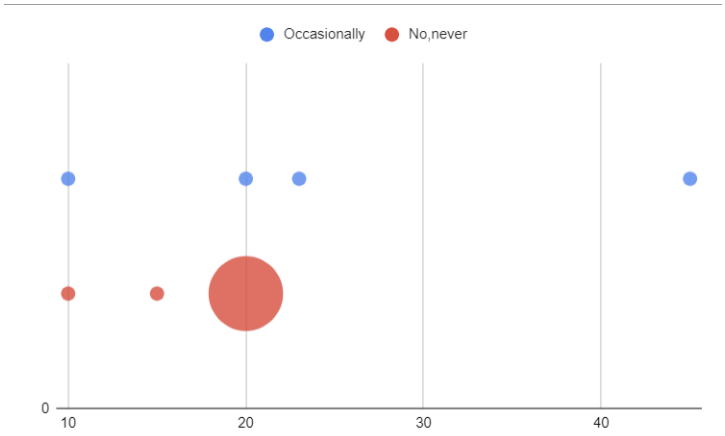

Figure 4.12: Bubble Chart depicting technical issues over duration

been reported in other VPL systems, such as **Max MSP**, as exemplified in Chapter 1 and Figure 2.2.

The absence of any discernible variation in the experience, relative to usage time of the application, across the other questions is likely attributable to the small sample size of participants and therefore data, which precludes the identification of any patterns. Nevertheless, the results described above shed light on the influence of usage duration on the users' experience. Specifically, a marginal deterioration in the experience was observed with an increase in time spent using the application.

4.4 Summary

Throughout the critical evaluation of this project, the user study produced was able to assess whether most of the design objectives were met for the final design of the web-application. The study showed that the application:

- Has succeeded in mitigating user disorientation and enhanced usability.
- Is more tailored for novice graphic designers compared to professionals.
- Is not able to measure if the poster design process was iterative and incremental
- Has partially succeeded in showing that the poster design process was novel, however the small sample size of proficient graphic designers makes the results uncertain
- Is less effective at portraying concepts more intuitively over Icon-based user interfaces
- Has a minor hindrance in usability the longer it is used

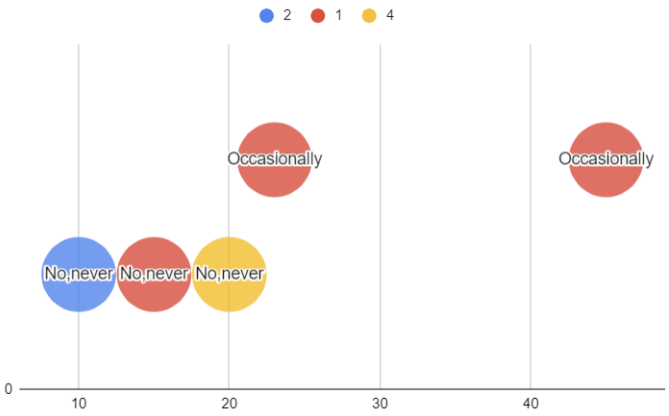

Figure 4.13: Bubble Chart depicting how often users feel overwhelmed by UI

Furthermore, the results of the user study are subject to certain limitations. Although the evaluation was planned to involve a larger and more diverse participant pool, the data collection was hampered by time constraints that limited the duration for which individuals could participate in the study. Moreover, based on the insights obtained from the study outcomes, it would have been preferable to pose additional inquiries to ascertain whether all project objectives were accomplished.

The full list of questions and answers from the user study can be found in the appendix.

Chapter 5

Conclusion

To conclude, a summary of the accomplishments of the project will be presented, along with the current status of the application and a critique of the conducted user study. Additionally, offering future plans for the project will be outlined.

Throughout the course of this project, I conducted research on various topics, such as Visual Programming Languages, Flow-Based Programming, and Poster and Graphic Design. This involved providing comprehensive definitions and identifying areas that pertain to the web-application of this project. Furthermore, I conducted a review of the relevant programs and assessed their strengths and limitations for my intended application.

In addition to conducting research, I designed and implemented three iterations of a Flow-Based VPL Poster Design application. The initial iteration served as a proof of concept to gain familiarity with web-application development and the Graphics API, which was utilised in subsequent iterations. Although the development of the first prototype was discontinued prior to achieving its design objectives, it provided a foundation for the second prototype. For the second iteration I designed and built a minimum viable product, comprising of a web-application that introduced new technologies such as React and Rete.js into the application. It consisted of incorporating a new Flow-Based VPL for Poster Design, whilst also providing a VPE to render the posters and provide direct manipulation of visual components to help with formatting the posters. The final iteration extends the code base of the previous prototype, refining the design with the consideration of the intended end-users. The final product expands the VPL's functionality through the development of new components that enable the creation of multimodal and able to be animated visual components within the poster design. Alongside expanding the VPE with the implementation of layers.

After producing the final iteration of the application, I built and conducted a user study for it. The study assessed the effects of the VPL on the user's design process, the application's usability, and the impact of usage duration on the user experience. Using the results, it was possible to perform an analysis to determine whether the final design had achieved its objectives.

Having arrived at this stage in the project, the attention will now turn to the present state of the project to assess the extent to which the initial goals and objectives have been met.

By means of the critical evaluation, I was able to ascertain if the application had accomplished its original objectives as set forth in Chapter 1. It was revealed that the target user group was only partially achieved. Rather than catering to both proficient graphic designers and novice ones, the application was more favourably received by novices than by experts. This disparity can be attributed to the limited features provided compared to other existing applications. In addition, the second segment of the intended user base, i.e., expert programmers, could not be adequately represented in the user study owing to inadequate participation. Thus, it was not feasible to determine whether the application was suitable for expert programmers or not. The application was successful in presenting a unique approach to the Poster Design process. In its current state the application is successfully widely accessible through being a deployed website. Furthermore the study shows that the user interface is easy to use, however not as effective as other approaches for depicting Poster Design concepts intuitively.

Despite the application's overall success in achieving most of its objectives, there are still several features that require further refinement. In the context of the VPL, specific attributes could not be integrated within the allotted time frame. Within the *Text* component, it would have been desirable to finalize the Point Cloud output, which is currently non-functional, as well as to include typographical features that are available in other similar applications. For the *Noise* component, it would of been

desirable to make the output connectable with the mathematics components to integrate it more within the VPL. Additionally, it should be noted that the *Blockify* component requires the *Noise* component to be functional within the VPL, unlike the other visual components that can operate independently. This deviation from its original design was due to time constraints that prevented full implementation of this component. Moreover, I had intended to add more input ports to component controls, specifically including positional input controls for the visual components. The absence of this feature limits the possibilities for poster animation.

Taking a broader perspective and considering the entirety of the application, there are certain unresolved issues and forthcoming plans. Presently, the application lacks export capabilities for videos, which poses a persistent challenge since animation features are available. Additionally, in order to appeal to the target user base of expert graphic designers, augmenting the VPL with additional components will be necessary. However, this may make the *Context Menu* difficult to use unless organizational features such as drop-down sub menus are integrated. In terms of the VPL, there are plans to include a series of Vector Graphics, sound and sound analysis components to offer a more comprehensive set of features. Moreover, as discussed in Chapter 2, utilising the FBP approach enables compounding of components, which can resolve the issue of overwhelming the user interface as the poster designs become more complex, as was observed in the user study. As the complexity of the user's posters increases and multiple sessions are required to complete them, the issue of saving works-in-progress arises. To address this, the website will need to incorporate a more extensive set of server-side functionalities for storing these designs.

In conclusion, although the application may lack certain features present in other graphic design software, the comprehensive investigation, development, and evaluation of this application has demonstrated that the utilisation of a Flow-Based Visual Programming Language is a promising and innovative approach to Poster Graphic Design.

Bibliography

[1] Adam. Building a website with gpt-4: From reference image to finished design, Mar 2023.

[2] Ivanna Attié. Adobe stats 2023 - all relevant numbers including adobe stock, Jan 2023.

[3] Marat Boshernitsan and Michael S. Downes. Visual programming languages: a survey. Technical Report UCB/CSD-04-1368, EECS Department, University of California, Berkeley, Dec 2004.

[4] Robert Burton. Ivan sutherland - a.m. turing award laureate.

[5] T. O. Ellis, J. F. Heafner, and W. L. Sibley. *The Grail Project: An Experiment in Man-Machine Communications*. RAND Corporation, Santa Monica, CA, 1969.

[6] Unreal Engine. Unreal engine 4.27 documentation: Blueprint compiler overview.

[7] Interaction Design Foundation. What is graphic design?, Sep 2021.

[8] Richard Frost. High-performance visual programming environments. *ACM SIGGRAPH Computer Graphics*, 29(2):45–48, 1995.

[9] Viktor Krakovsky. Software development models: Iterative and incremental development, 2014. Accessed on: May 5, 2023.

[10] Geunhee Lee and Iis P. Tussyadiah. Textual and visual information in ewom: A gap between preferences in information search and diffusion. *Information Technology amp; Tourism*, 12(4):351–361, 2010.

[11] Vladimir Lugovsky. Council post: A guide to low-code/no-code development platforms in 2021, Sep 2021.

[12] R. Mahajan and B. Shneiderman. Visual and textual consistency checking tools for graphical user interfaces. *IEEE Transactions on Software Engineering*, 23(11):722–735, 1997.

[13] Unity Manual. Refactor a c script with visual scripting: Visual scripting: 1.7.8.

[14] J Paul Morrison. Flow-based programming :: Comparison, 2011.

[15] J Paul Morrison. Flow-based programming :: History, 2011.

[16] Yvonne Rogers. Hci theory: Classical, modern, and contemporary. *Synthesis Lectures on Human-Centered Informatics*, 5:1–129, 05 2012.

[17] Paul Shaw. Paul shaw letter design " the definitive dwiggins no. 81a-w.a. dwiggins and "graphic design": A brief rejoinder to steven heller and bruce kennett, May 2020.

[18] Ben Shneiderman. Direct manipulation: A step beyond programming languages. *Computer*, 16:57–69, 1983.

[19] Sarah Walker. Research in graphic design. *The Design Journal*, 20(5):549–559, 2017.

[20] Kirsten N. Whitley and Alan F. Blackwell. Visual programming: the outlook from academia and industry. In *Workshop on Empirical Studies of Programmers*, 1997.

Appendix A

Appendix

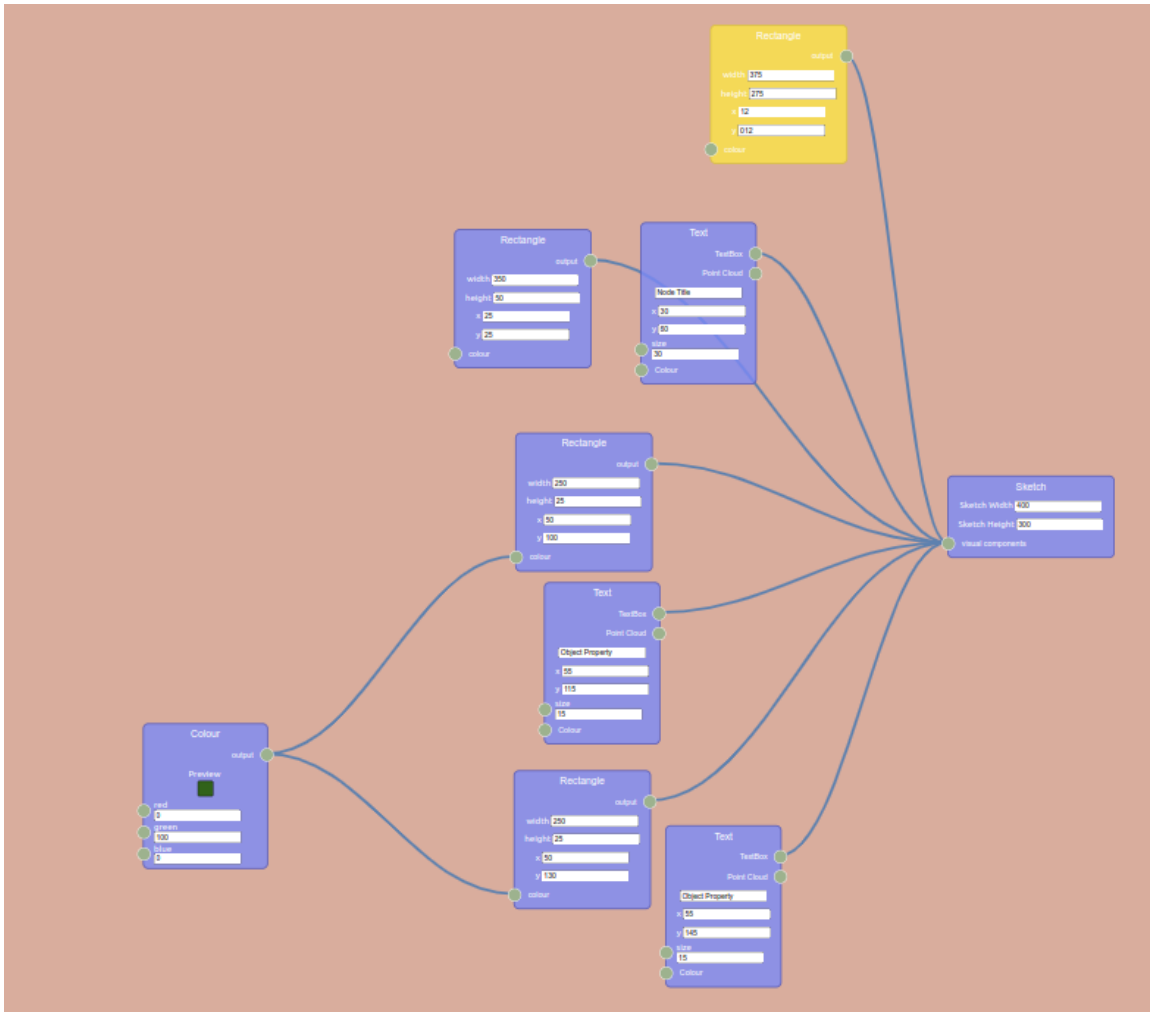

Figure A.1: VPL Editor for Figure 3.1, using Prototype II

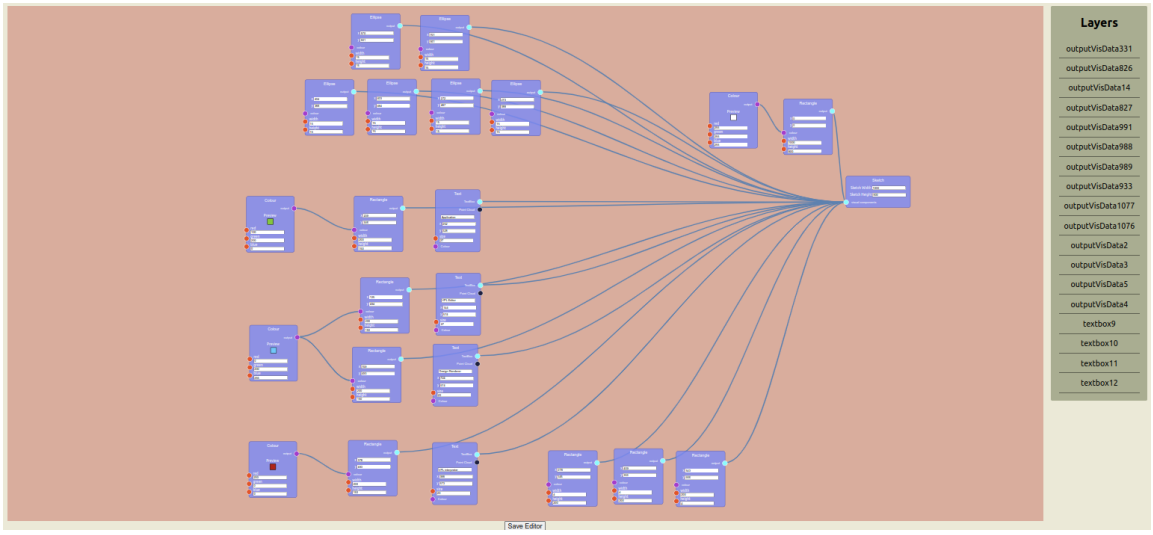

Figure A.2: VPL Editor for Figure 3.5, using Final Product

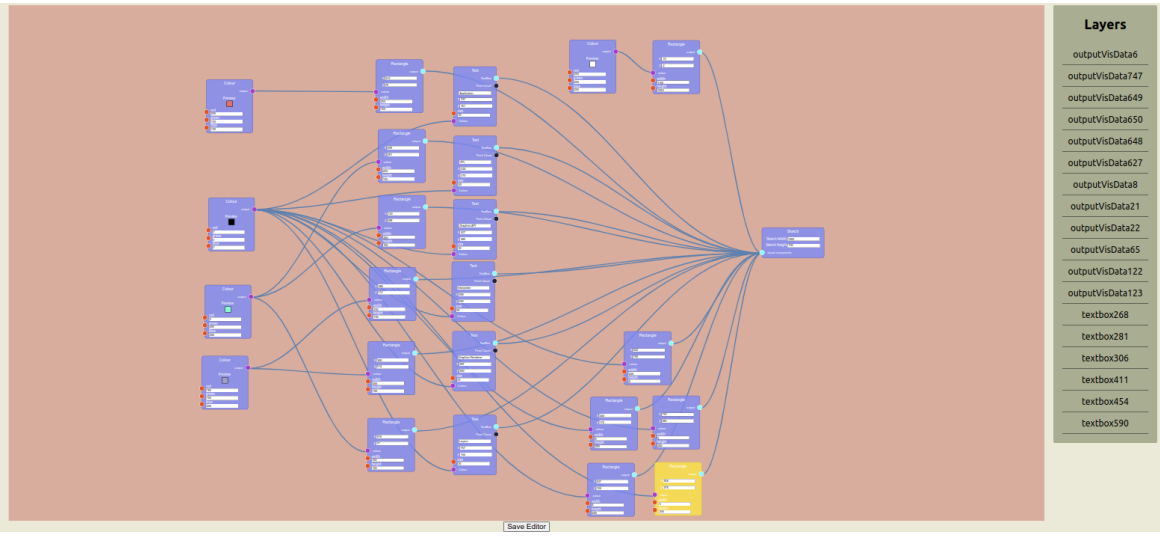

Figure A.3: VPL Editor for Figure 3.10, using Final Product

Figure A.4: User Study Questionnaire Page 1

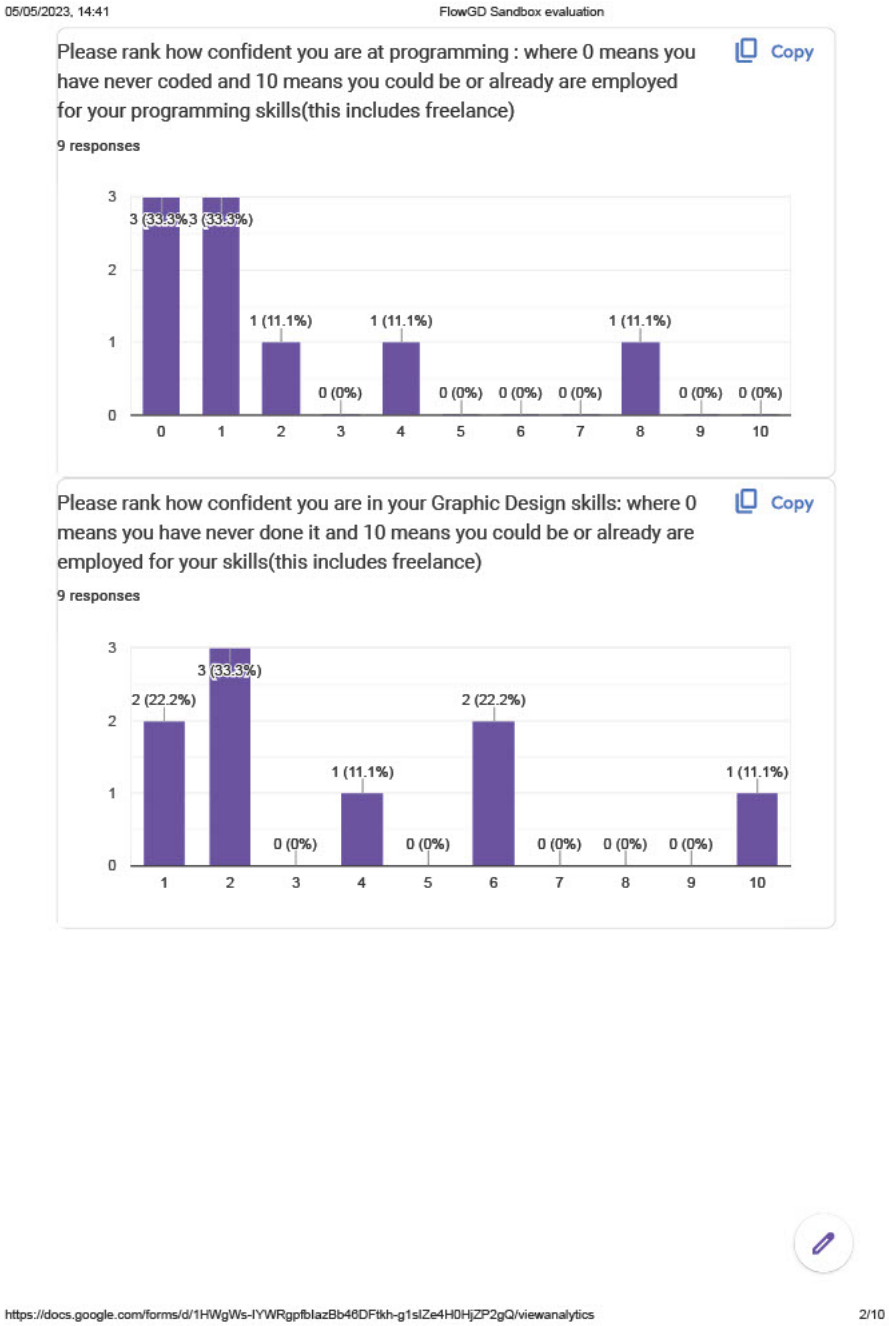

Figure A.5: User Study Questionnaire Page 2

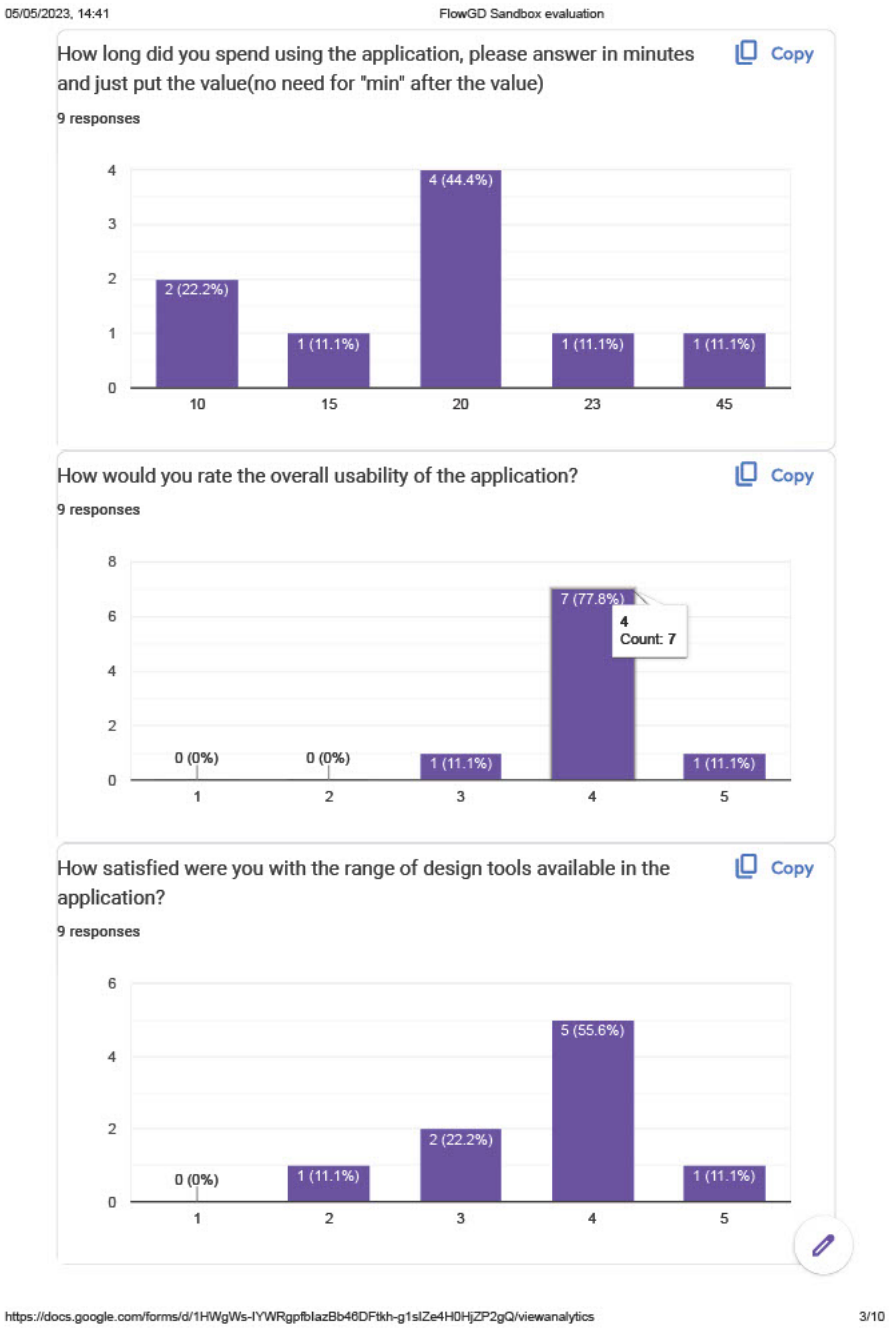

Figure A.6: User Study Questionnaire Page 3

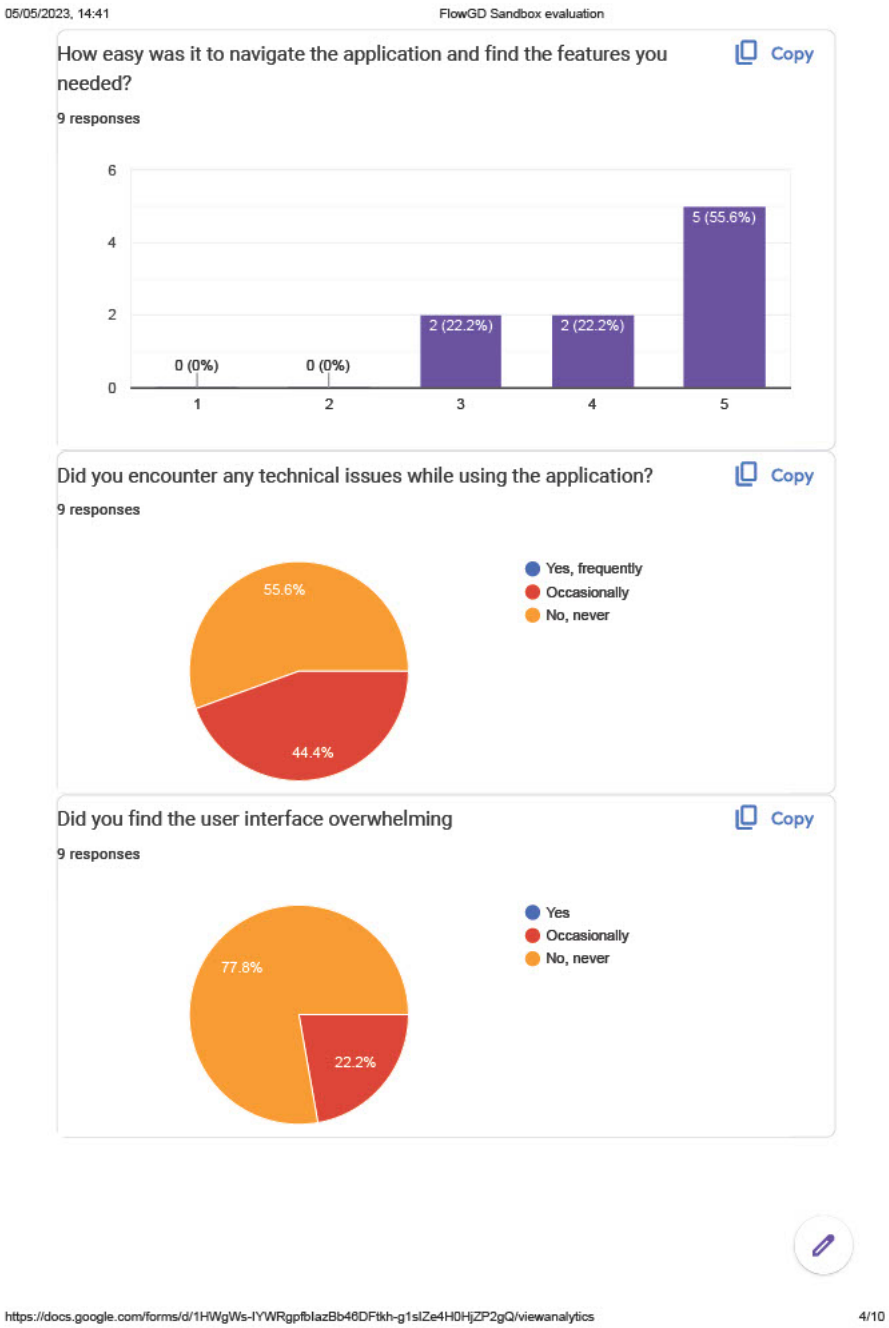

Figure A.7: User Study Questionnaire Page 4

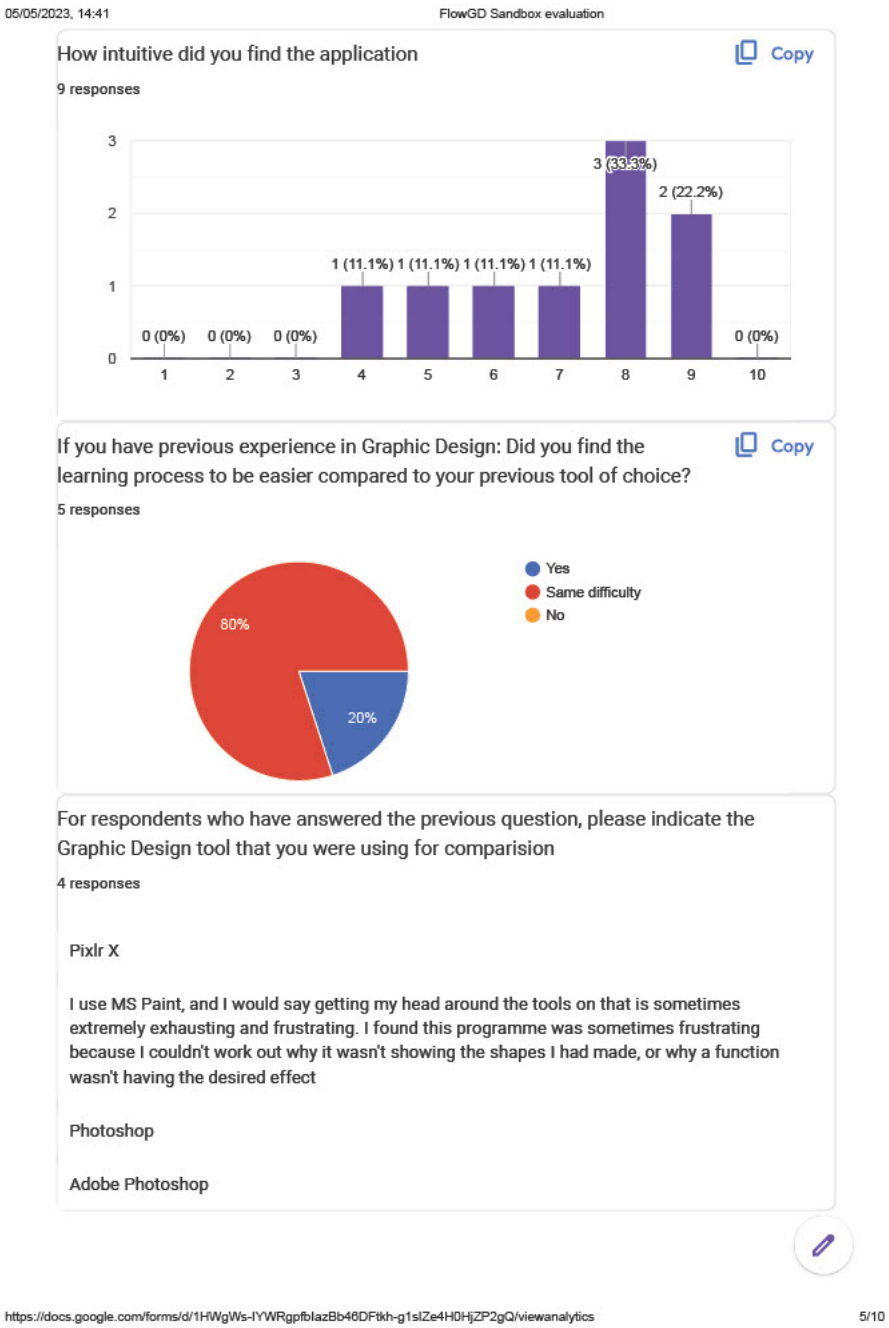

Figure A.8: User Study Questionnaire Page 5

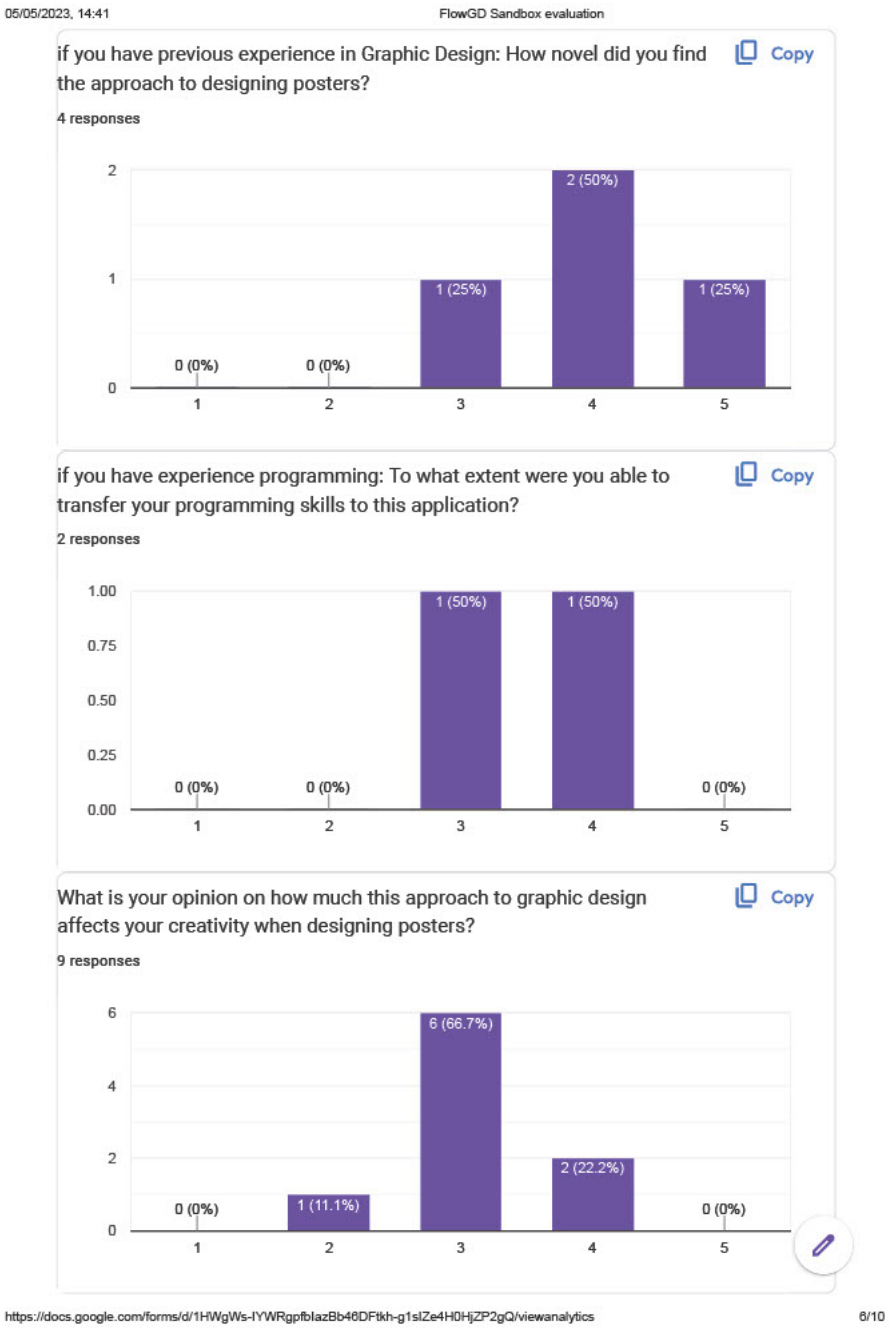

Figure A.9: User Study Questionnaire Page 6

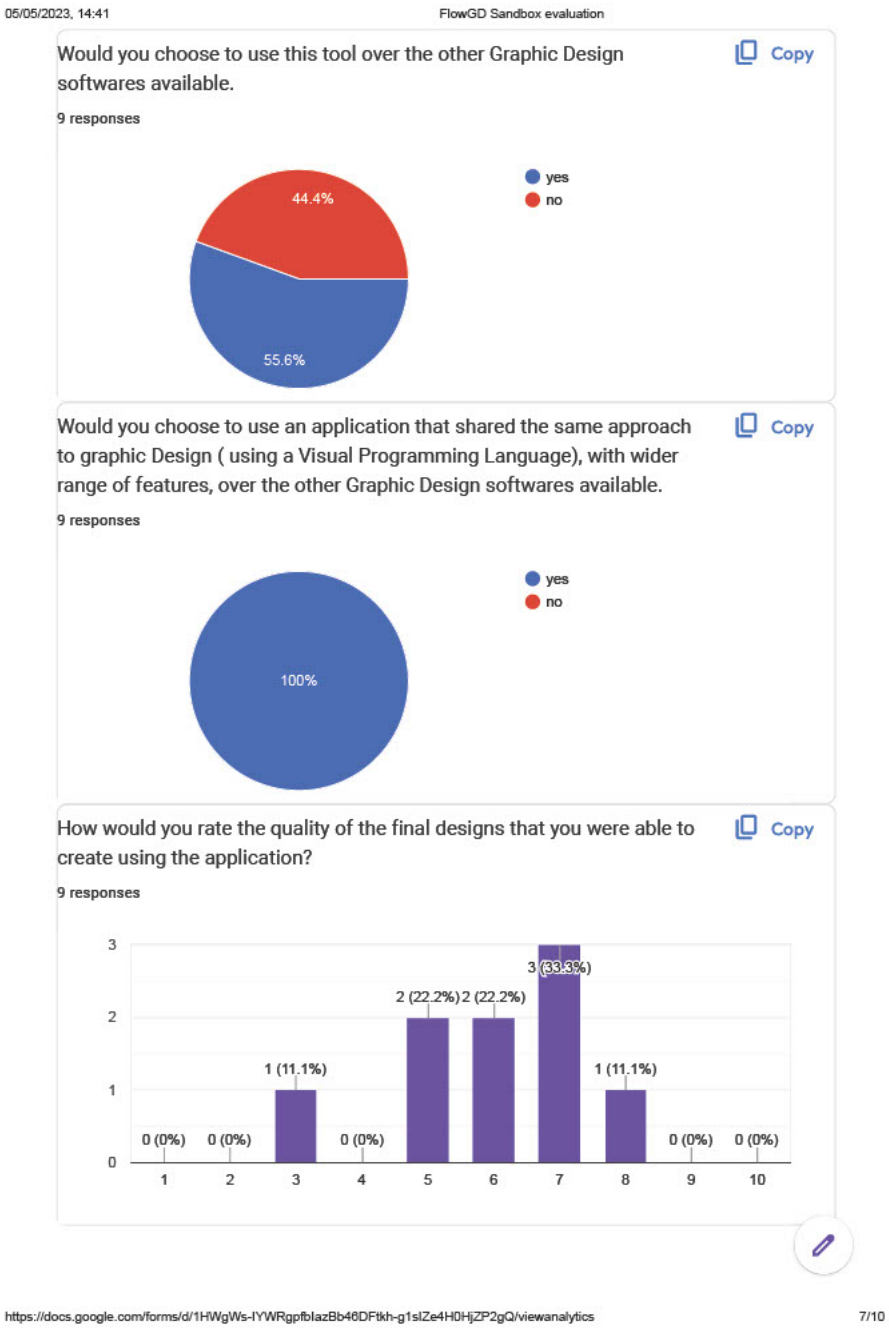

Figure A.10: User Study Questionnaire Page 7

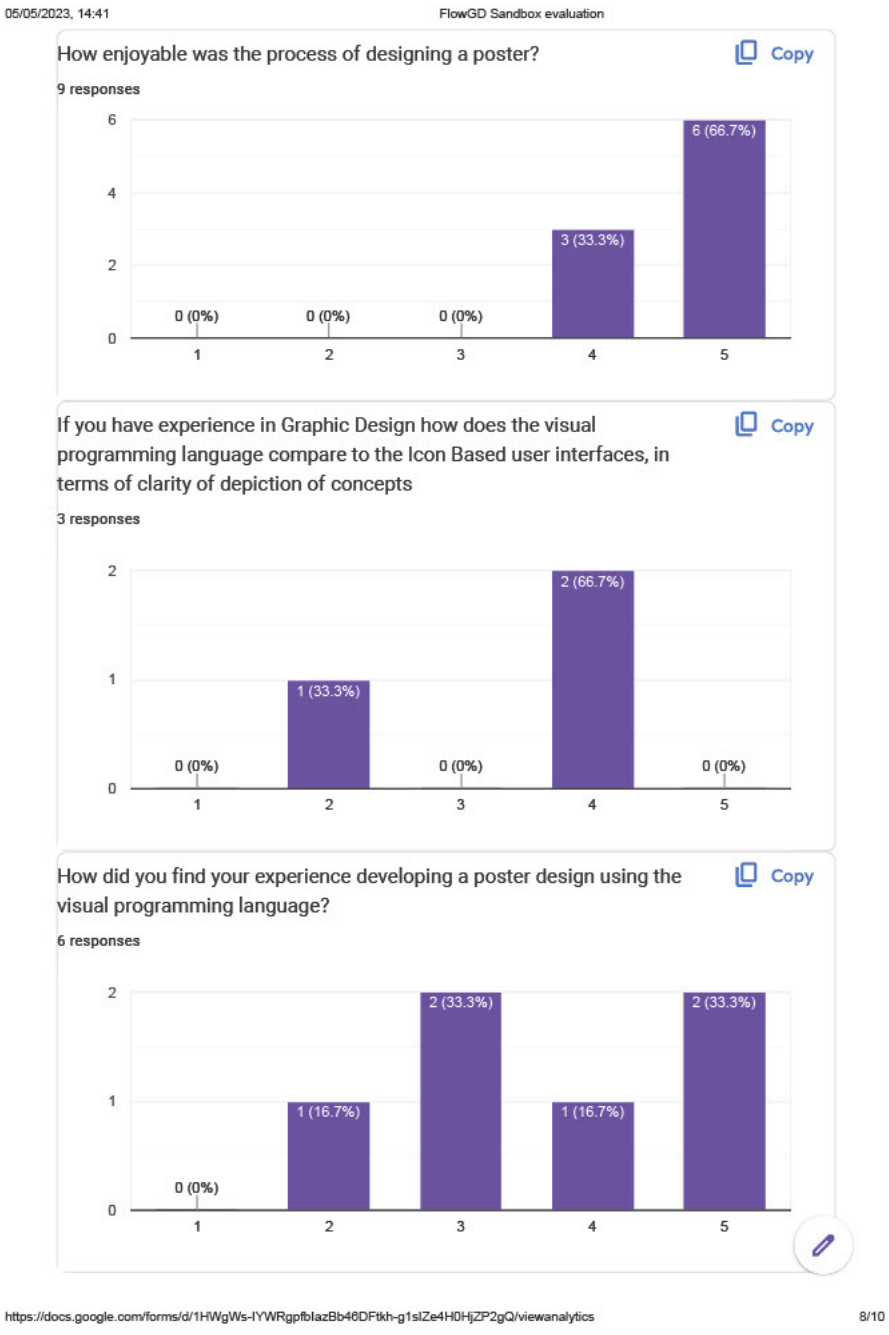

Figure A.11: User Study Questionnaire Page 8

Figure A.12: User Study Questionnaire Page 9