Prifysgol Cymru
Y Drindod Dewi Sant
University of Wales
Trinity Saint David

BSc Degree in Software Engineering
2021-2022

BSc Thesis

“VisualPro: Research and Improvements of Junior Software Engineering”

Edward Samuel Ralph Patch

Mike Dacey
Waterfront Campus - 2021

SUMMARY

Keywords: Visual Programming, Visual Scripting, Development

DEDICATION

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

VisualPro is a lightweight, high-performance Visual Scripting client that enables users to create high-end software, whether the logic is for, not limited to, web, software, Artificial Intelligence. VisualPro would extend the Programming Planner library to show off the potential of the usage and potential of the library, in turn, bringing a different design of Visual Scripting to the table. The software aims to increase workflow without discriminating cognitive ability, whether it is struggling to adapt to a different language or acquiring new skills that impact how the software works.

VisualPro aims to unify languages to achieve, not only for users to adapt to complicated languages but also guide the user to learn what languages have in common and therefore increase the drive to learning new languages without feeling 'there is only one tool for the job' condition.

Features that VisualPro brings forward to the table are as follows:-

- **Novice Development:** Studies of how beginners learn and how VisualPro can improve.
- **Dynamic Languages:** The ability to add new languages, the code generation can end up as.
- **Visual Scripting Software:** A tool with zero restrictions on what code the user wishes to generate.

Requirement steps for the VisualPro program include rewriting the Programming Planner console application, written in the C++ language to work in dynamic-link libraries and supporting dynamic languages. After the previous stage, designing the VisualPro interface and learning how a beginner may use the software are essential. Ultimately, implementing the product that achieves a Visual Scripting program that is lightweight, generates quality code and is easy to use for beginners with the potential of persuasion of intermediate and advanced programmers.

2. LITERATURE REVIEW

2.1. User Interface and User Experience

Research for User Interface (UI) and User Experience (UX) analyse how a new developer would use the software with little to no coding experience and what difficulties the UI and UX design may encounter. Methods of outlining and testing the UI and UX are within the chapter, Research Methodology 3, page 12.

2.1.1. User Interface

To get some ideas for the UI, the Journal 'Evaluation of a UML-Based Versus an IEC 61131-3-Based Software Engineering Approach for Teaching PLC Programming' writting by Birgit Vogel-Heuser, Martin Obermeier, Steven Braun, Kerstin Sommer, Fabian Jobst, and Karin Schweizer [1] and the Journal 'Design Issues and First Experiences with a Visual Database Editor for the extended NF^2-Data Model', author(s):- K. K*ü*spert, J. Teuhola and L. Wegner [2]. These Journals cover Unified Modeling Language (UML). The UML concept is promdomity used in relationship diagrams in databases and Visual Scripting.

The addition of UML tools for the frontend to create the Visual Scripting platform proves an excellent opportunity as there is plenty of UML Framework support. Some critical points based on an experiment from Birgit Vogel-Heuser's Journal Article [1] are listed.

To gather the answer for 'How effective is UML tools?', the hypothesis from Birgit's [1] Journal article answers the question by saying, "Students trained in OO modelling show an improved modelling performance.".

The experiment environment involves the following rules:-

- Software Engineering theme. - Any material created for the experiment to work, the experiment outline is set in a Software Engineering equivalent environment.
- Training Assets and Problem-Solving Exercises. - A series of training materials and problem-solving exercises were created for the experiment to be fair.
- Task Setting and Training. - The author focuses on Task Setting and Training, supported by Hierarchical Task Analysis.

To answer Hypothesis 1 set by Birgit Vogel-Heuser [1] '...analysis examined the within factor (before and after the training) and additionally two between factors (notation and expertise)", the extract as follows is chosen.

"The data were analyzed using three methods. First, an analysis of variance (ANOVA) was applied to test differences between the variances of several groups in order to show whether their performance changed after the training approaches and whether their performance differed between classes due to expertise level differences. Then, correlations were computed as a measure of the relation between programming/modeling performances on different performance scales. Subsequently, differences in relations between the two software engineering approaches (see hypothesis 3) were shown by computing regression models for the programming performance for both approaches." - Birgit Vogel-Heuser [1].

The results gathered display that the training set for learning UML in a Software Engineering sense was highly effective. However, the Birgit Vogel-Heuser [1] states that 'All participants learned from the training.' and backs it up by '...indicating that the - group had learned even more from the training because of their poor results before the training (see also Fig. 2) and their lack of prior knowledge.' This quotation suggests that UML /tools found in Visual Scripting may not necessarily be complicated for beginners to learn, but they may seem more complex without proper training. This conclusion could suggest that UML methods are not well documented and could indicate that UML may not be well maintained, look at section 2.1.2, page 3.

2.1.2. Maintenance

The Journal by Erik Arisholm, Lionel C. Briand, Siw Elisabeth Hove and Yvan Labiche, 'The Impact of UML Documentation on Software Maintenance: An Experimental Evaluation', addresses the lack of UML Documentation on both developer-side and client-side. After learning that the individuals who participated in the experiment from section 2.1.1, page 2, the question arose, 'Was the UML documentation depreciated and how was the documentation withstanding on the development side?' it was vital to determine if UML tools are future-proof or will it prove expensive to maintain. After Erik Arisholm performed tests found in the Journal, the author found that 'UML documentation does not seem to provide an advantage when considering the additional time needed to modify models.' Furthermore, disregarding existing UML documentation seemed to correlate compared to the first test, thus suggesting that the UML documentation had the same effect as no UML documentation.

Things to note during this study, if the integration of UML tools within VisualPro, then expenses regarding finance, staff and resources for UML documentation for both developers and clients are necessary. However, the creation and maintenance of the software would require UML Development documentation. Yet, reverse engineering the UML li-

braries to create a detailed UML Development and UML Usability Documentation for both users before any development can begin on the software. This event is due to understanding how the UML tools operate and integrating the software properly to work in the future.

2.1.3. User Experience

Within the Journal, written by Birgit Vogel-Heuser, Martin Obermeier, Steven Braun, Kerstin Sommer, Fabian Jobst, and Karin Schweizer [1], 'Evaluation of a UML-Based Versus an IEC 61131-3-Based Software Engineering Approach for Teaching PLC Programming', an experiment to test whether beginners excelled at Object-Oriented Programming (OOP) or Functional Programming (FP). The quote follows:-

"Berges and Hubwieser investigated Computer Science freshmen's abilities to learn OO programming in two and a half days with as little (human) instruction as possible [27]. Examining 300 students' program code, they found that most were able to write quite satisfying programs. They identified two types of students: those who accept and apply the OO concepts, and those who prefer to program in a more traditional procedural way. They also tried to define the characteristics of object orientation to evaluate measures for program quality, e.g., one instance of a class is created." [1]

This test suggests that after three hundred students were tested with little instruction from other peers, which opens an interesting fact that most of the three hundred students wrote acceptable code in both OOP and FP categories. The test's conclusion finds two types of students, and the students either found OOP or FP easier. After analysing this specific test, beginners may have different mindsets and thinking styles.

2.2. User Usability

As the principle of the product covers beginners' use of Visual Scripting to aid the users with the proficiency and capability to understand and further their logic building skills, it is vital to understand how usability is measured and evaluated. According to Dana Chisnell [3] Usefulness, Efficiency, Effectiveness, Learnability and Satisfaction are five necessary components of how the usability tests should carry out. These selections establish how the target audience will interact with the software and test if the target audience stays interested in the software without feeling any frustration towards the software.

VisualPro's end product needs to increase the efficiency of standard programming and Visual Scripting and manage to keep effectiveness and usefulness. For example, based on Dana Chisnell [3], the software could achieve the '...system is easy to use, easy to learn...' and is '...satisfying to use...', the product will gain no interest and have limited use if ignoring these objectives. This example backs up that it could be catastrophic with too

much focus on one topic. However, if the software generates the information for the target audience and still keeps the audience interested, it could change the Web and Software industry as we know it today.

Detlef Zuehlke [4] mentions that using a survey and final testing of user usability is not efficient to guarantee a usable UI. As VisualPro intends to create a more effortless Programming experience, other usability testing methods are required.

A method of deciphering the results from any quantative research from User Usability testing is to group the results using Fuzzy Logic that enables results to fall into different categories to convert to a binary format, for example, *1* being effective and *0* being ineffective. Fuzzy Logic enables two sets of data with multiple thresholds to combine to create a noisy environment into a more straightforward process.

Extensive research discovers how and if Fuzzy Logic is an excellent tool to examine and conclude any usability tests. The Journal Article, Marwa Hentati [5], 'A Fuzzy-logic System for the User Interface Usability Measurement' mentions efficient ways of how Fuzzy Logic help with the conclusion of Usability Testing and provides examples of measurement references to create the best analyst. Marwa Hentati [5] found that an ideal fuzzy system uses three main contributions to produce effective use of results found from User Usability testing.

2.3. Productivity and Visual Scripting

Productivity, specifically within the software engineering field, Dr. Caitlin Sadowski's [6], 'Rethinking Productivity in Software Engineering' Book states, "Productivity is a challenging concept to define, describe, and measure for any knowledge work that involves nonroutine creative tasks. Software development is a prime example of knowledge work, as it too often involves poorly defined tasks relying on extensive collaborative and creative endeavors. As in other areas of knowledge work, defining productivity in software development has been a challenge facing both researchers and practitioners who may want to understand and improve it by introducing new tools or processes." When reading further into the chapter, the following points evaluate how we should think about productivity:-

- "Velocity: How fast works gets done." [6]
- "Quality: How well works gets done." [6]
- "Satisfaction: How satisfying the work is." [6]

After thinking about this, VisualPro, in theory, should increase the rapidity of the individual's work. Suppose VisualPro changes the way Visual Scripting works today, making it lightweight and understandable compared to existing Visual Scripting (For example, Unreal Engine's Blueprints [7]), with the ability to generate any desired code. In that case, the quality and satisfaction of the developer amplify.

2.4. Learning Tools

Research of obstacles and potentials help identify existing and future issues that VisualPro software could generate. The aim is for amateurs new to the development field and suggests existing problems that put a 'brick wall' in learning programming and potentials of how VisualPro can solve these issues. This topic may reflect on the critical components of UI and UX previously mentioned.

2.4.1. Learning Styles

After observing Tileston Donna Walker's [8] book, for VisualPro to increase programming skills to beginners, three styles, *visual, auditory and literature*, benefits any implementation with the software or documentation. It is essential not to overuse any style to pursue a healthy balance with this software, as some individuals like a challenge to overcome problems before being presented with a solution.

2.4.2. Interactive Tutorials

"In EE 220, Network Analysis I, 12 tutorials were created. One of these takes the student through the solution of a three-phase power problem. First, the student receives a statement of the problem and a diagram of the circuit. The student is then asked to find the power consumed in one phase of the load. If the student answers incorrectly, a hint is given: "One third of the power is used in each of the three phases of the load." More hints are given until the student gets the right answer. The student then moves on to the next page of the lesson, asking for a calculation of the line voltage. After 8 pages, the student has solved the problem for all the voltage, current, and power values of the circuit" - Dr Thomas G. Cleaver [9].

Dr Thomas G. Cleaver set up the Network Analysis I tutorial experiment, creating 12 tutorials with different properties. A particular property is a hint system that would work well in VisualPro's tutorial system. As previously described, not all individual's likes step-through guides and may prefer to work in a challenging environment. A solution based on this example tutorial provides a hint to the user.

From the understanding of Dr Thomas G. Cleaver [9] Interactive Web-Based Tutorials for Engineering Education, a feature from the 'RAISE (Remote Asynchronous Instruction in Science and Engineering) program at the University of Louisville...' that enables individual's to create interactive tutorial's with 'rudimentary knowledge of HTML'. Based on this concept, separate interactive tutorial's could enhance the learning process for beginners using VisualPro, or better yet, the adaptation of interactive tutorial's could work within the VisualPro's environment. A plan to create an interactive tutorial is to design tutorial files that create an end product that will load a series of highlights over draggable

items and text areas with a box that flags the user with the entry data. With relevant tips and examples during each interactive tutorial pack should excel the learning process.

2.4.3. Technical Tutorials

The Journal Article, 'Three Levels of Guidance in Technical tutorials', written by Robert Krull [10], suggests some key points to consider when making a documented tutorial. Robert Krull [10] hints that most technical tutorials work best under supervision from the quote 'At Level 1, the most structured of the tutorial approaches considered in this paper, learners work under the strong guidance of a coach.' It is important to expect that novice users' of VisualPro may trip up if following the technical tutorials alone. Robert Krull [10] suggests controlling the learning process within the Technical Tutorial to help guide the audience rather than provide an unstructured tutorial. When teaching the audience, it is crucial not to '...criticize their efforts with feedback that labels their efforts right or wrong...' [10], especially as the fault could lay in different aspects like UX and User Usability. The last point, Robert Krull [10] mentions that repetition should exist in tutorials to help the audience learn the process, rather than creating unwanted confusion. The tutorial could use a consistent house style to read, try an example, and 'try it yourself' methods to keep the repetition going. This repetition will gain accurate feedback to test UX, User Usability and see how novices respond VisualPro. Consistent repetition will increase the audience's morale.

2.5. Tools for the User Interface

Whilst observing the book, 'Using UML: software engineering with objects and components, by Pooley R. J. and Perdita Stevens[11], it is important to think about why UML may make a good UI method. UML intends to meet the criteria of allowing Software Engineers to design a software platform and demonstrate it to their clients. At the same time, it gives a comprehensive plan instead of focusing on Pseudo code, supported by the quotes, Pooley R. J. [11] '...help to resolve misunderstandings' and '...developer's effort can be spent on work that requires their skills, not on routine work such as making a diagram using a drawing tool;'.

Another method of creating the UI is a Node Tree that displays the objects in containers and links them together with arrows to display the object's relationship. A Visual Database Editor diagram in figure 2.1 illustrated by K. K*ü*spert [2]. The node tree, in this case, is to construct a database structure design. Although, this could prove an easy-to-read method for Visual Scripting. Perhaps a mix of UML and Node Tree's could benefit from displaying relationships, for example:- *Parents* → *Children*. In theory, VisualPro could potentially change Visual Scripting as it is today and demonstrate the power of the C++ Dynamic-Link Library (DLL), Programming Planner library can do.

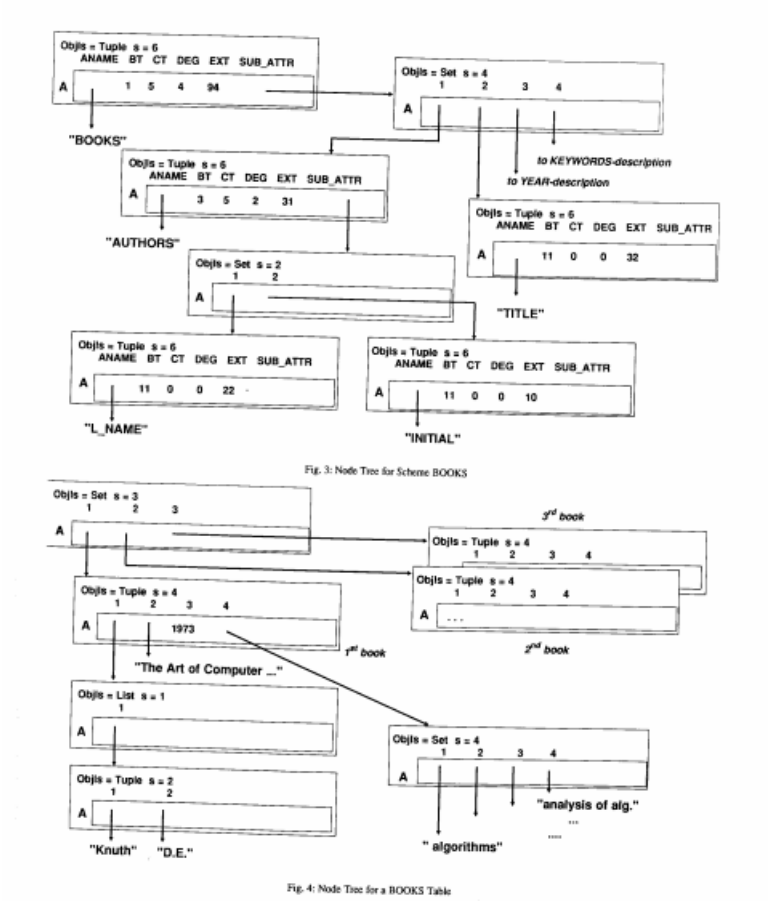

Fig. 2.1. Visual Database Editor Diagram [2]

2.6. Libraries for Functionality

From GitHub Repositories, a list of Visual Scripting libraries are listed:-

- Language: GDScript - Author: Swarnimarun - Respository: Visual Scripting Node Library.
- Language: Python - Author: leon-thomm - Respository: Ryven.
- Language: C# - Unity Game Engine - Author: ConstellationLanguage - Respository: Constellation.
- Language: Web Scripting - Author: ericabouaf - Respository: webhookit.
- Language: Web Scripting - Author: WebCabin - Respository: wcPlay.

These Visual Scripting software listed are identical to standard Game Engine Visual Scripting logic. Even though some are easy to use in the demos, a commonality is that they only allow users to control a live environment, whether it is a built-in game engine or an addition to an existing Game Engine, and they are mainly game oriented. VisualPro aims

to support different platforms, such as Web, Mobile, Software, Data Analytics and Games. This aim is possible due to the diversity of users, adding new languages by entering the desired language within the XML document; though, it requires documentation to support users in adding new languages for the program to work correctly.

A table of a few Web Builders and Visual Scripters demonstrates the difficulty of different platforms. Visual Scripting Software 2.1:-

Name	Description	Difficulty: Hard (0-10) Easy	Explanation?
Unreal Engine 5 [7]	Game Engine (C++ Language)	Beginner: 3-0 | Experience: 5-10	A beginner who does not understand the principles of code may struggle over time.
Unity Engine [12]	Game Engine (C# Language)	Beginner: 2 - 6 | Experience: 8-10	With the documentation available and the simplicity of the Unity Visual Scripting design, it seems beginner-friendly.
Minecraft [13]	Game (Redstone) Represents binary coding.	Beginner: 7-10 | Experience: 10	Even though Minecraft does not create programming languages, it shows that many ages who enjoy logging onto Minecraft to make Redstone functionality, passively learn about Binary code.

TABLE 2.1. COMPARISON OF VISUAL SCRIPTERS

After studying different Visual Scripting software, there is no portable Visual Scripting made for versatility. Both seem to only aim at Game Engines. Minecraft does not offer the ability for Visual Scripting. However, to bring attention to the Redstone feature in Minecraft, the feature is interesting for the audience, even to none-programmers. Could the VisualPro library be made into a Visual Scripting game, or specific in-game blueprints/prefabs be placed into the environment, generating code, so that the user can create programs whilst maintaining interest? Redstone accomplishes Binary teaching with Redstone passively by using on and off instead of 1s and 0s.

Visual Scripting, Unreal Engine has a 'messy' look, especially when the game logic is more clunky and complex, whereas Unity seems to have a better order. The problem with Visual Scripting in both engines is that the performance of the code is slow compared to writing the code manually. Would this uplift the software market sections if the VisualPro Scripting Pad software could generate code in the desired language with the most effective running time?

In one form, Web Builder's give an idea of what makes Visual Scripting easier. This concept is down to the research, design, and implementation of existing developers who had to look into individuals with no coding experience. If there is a correlation between designing a more straightforward Visual Scripting interface using methods that Web Builders introduced, it could update Visual Scripting and bring it into future software design. Perhaps, it could be a tool to visualise JavaScript so that users can build a website and add functionality, or create code snippets, for example, MelPy to animate 3D models within Autodesk Maya and other products, making Artists thrive more rather than

Name	Description	Difficulty: Hard (0-10) Easy	Explanation?
WordPress	Web Builder	Beginner: 8-10 | Experience: 9-10	WordPress offers many themes and plugins to the user base.
Wix	Web Builder	Beginner: 9-10 | Experience: 10	Wix is designed to help buisnesses to design webpages with no skill required.
Bootstrap Studio	Web Builder	Beginner: 5-7 | Experience: 7-10	Bootstrap Studio is a similar interface to Adobe Dreamweaver. It offers templates and allowed websites to designed. This interface is difficult with no experience with Bootstrap.

TABLE 2.2. COMPARISON OF WEB BUILDERS

relying on a Software Engineer to make their vision.

2.7. Testing Functionality

Testing functionality, whether performance, bugs or critical issues, is essential, especially before conducting any User Usability tests for many different reasons. The following scenario should create an understanding of why this stage is critical.

> "Unit tests not only catch implementation bugs, but also have a positive effect on the design of the code under test (CUT) as they enforce testability [3]. Code developed without unit tests in place often has a much lower testability, which in turn increases the cost of retrofitting unit tests later on." - Clause Klammer [14].

Trying to shape User Usability testing around Software with no unit testing creates a complex testability environment down the line. Clause Klammer's [14] states, 'Code developed without unit tests in place often has a much lower testability...' that suggests further testing like Performance testing and User Usability tests could prove costly if poorly written code is in play. Furthermore, the Journal Article [14] argues that code is hard to maintain and proves more challenging to evolve, backed up by 'As our experience taught us, the consequence is a heritage of legacy code that is hard to maintain and evolve.' Clause Klammer [14] raises the fact that after the implementation stage of the code, Unit Testing is rarely added and left alone, backed by 'However, over years of working with legacy code, we found that unit tests are rarely added afterwards. Code that has not been developed in conjunction with unit tests right at the beginning typically remains without unit tests ever after which conforms to Don Well's statement: *"If you do not start adding UnitTests today then one year from now you will still not have a good unit test suite." [2]*'.

Unit Testing is an excellent way of providing functionality testing, as a developer can create code directly from design plans before any implementation is committed or create and implement code around the test. From the Journal Article, Zenon Chaczko [15] suggests that unit testing indicates any flaws in the code and looks for any lousy programming practices. It is usually the case to pass a test before implementing the final product. Two unit tests are vital as the application uses both C# .NET Framework 4.7.2 frontend and C++ backend. This process aids the software to meet a standard and provides the right results for both the developer and tester.

3. RESEARCH METHODOLOGY

This chapter describes the different methodologies available, methods of analysing the results and constructs an experiment that would provide insight on how User Usability should commence. After extensive research, four techniques closely reflect on VisualPro's testing. Quantitative Research, Qualitative Research, Descriptive Research, and Fundamental Research help justify the techniques VisualPro intends to help novice programmers.

Quantitative Research enables calculations to provide answers to complex questions. This research method can measure productivity and User Usability to ensure the software is beginner-friendly. It is crucial to reflect on the calculations mentioned in the Literature Review to evaluate the data throughout the design period. **Qualitative Research** focuses on satisfaction experience, behaviours and intellect. A requirement of a complete analysis to check credibility. A survey inviting a mixed background of participants would support the research. **Descriptive Research** uses statistical data to analyse any design flaws within a project using official data. For accurate data that is relevant, the most up-to-date information dependent upon the scenario is suitable. **Fundamental Research** allows multiple research methods to correlate the information in a design plan. To justify the best research methodologies suitable for VisualPro's, Quantitative, Qualitative and Descriptive Research will support the design and development research of VisualPro. These methodologies help limit design flaws using up-to-date statistics, calculations to understand complications of comparing data to the gathered statistics, and any current design flaws within prototypes and VisualPro end implementation.

The data collection requires a survey, screen recording and microphone data of applicants. This extra data will coincide with the Preference side of the test to understand if the software enables beginners to learn quickly with excellent UX and User Usability.

To gather feedback, a technique of two surveys, two tutorial documentation and video recordings of users using the application. The first survey is to collect feedback on how the design of the prototype, 'Programming Planner', should feel if it had a UI applied to the software and suggestions of ways to improve the overall UX and User Usability of the software. This survey will prioritise *Fundamental Research*, collecting both *Qualitative* and *Descriptive Research* to get as much detail as possible to make a Visual Scripting software that proves effective and easy to use in the novice section of the Software Engineering. An interesting point to make is that the 'Don't Make Me Think, Revisited' book, authored by Steve Krug [16] suggests that 'Testing one user is 100 percent better than testing none.' and 'Testing one user early in the project is better than testing 50 near the end.' This suggestion is an ideal point to make as it suggests that early testing of the User Interface is a compelling idea to increase User Usability results during the creation period. To do this, the execution of the initial survey will test the prototype within

a console application, using more experienced programmers, and then test the VisualPro User Interface within the final survey with both novice programmers and initial survey participants to ensure the User Interface is intuitive and easy to use.

Two tutorials will give a temporary non-interactive learning environment to test the VisualPro in a learning environment for the second survey to work. Based on the research conducted, an interactive tutorial available inside VisualPro would help teach novice programmers by providing examples and other how-to exercises easier. However, having documented tutorials may benefit different learners who prefer a lightweight Visual Scripting pad without compulsory tutorial exercises and tips. Perhaps, a way to sort this problem is to provide a dialogue that asks the user three questions, 'Skip All Learning Plans', 'Interactive Learning' or 'Non-Interactive Learning', changing the start-up procedure. Tutorial-A and Tutorial-B provide participants in the final survey with insight into how to write code in the VisualPro tool and teach basic OOP and FP styles.

The second survey diverges into two sections, which are:-

- Programming Planner Improved vs VisualPro.
- Tutorial Questionnaire.

This survey aims to collect *Descriptive Research* to gather research on whether VisualPro increases workflow compared to Programming Planner Improved and if VisualPro delivers a learning and valuable tool to novices when learning about Visual Scripting or Programming in general. The 'Programming Planner Improved vs VisualPro' section is only applicable if the user has participated in the last survey. Within this section, the surveyee will answer a series of questions that reflect on their first answers to the first survey. Within the last section, all participants will observe and answer questions to test what they learned in the tutorial exercise and show their code generations. Results of whether the participant was successful, unclear or was unsuccessful are weighted, one being successful and zero being unsuccessful. If any generated code is on the right lines, they have a value of 1. If any input code is unclear, as in the copied code has any errors but the answers to the exercise questions are fitting, they have a value of 0.5. Otherwise, they are assigned the value of 0.

The final survey asks participants for video footage containing screen recordings using VisualPro Visual Scripting software; no requirement for the video or audio content. The footage provides an analytic opportunity to test if the software provides excellent UX and User Usability for novice and experienced programmers. This approach delivers Descriptive Research to encourage a comprehensive analysis of the UX and User Usability to enhance the quality of VisualPro's UI before forming the product further.

Examining the 'Don't Make Me Think, Revisited' book, authored by Steve Krug [16] justifies what type of results are a reasonable aim for collecting research. In the chapter "Chapter 9. Usability testing on 10 cents a day", the author highlights that '...Focus

groups are not usability tests.' indicating that whilst gathering information on whether the participants could generate code with little knowledge, it does not make it an easy-to-use and idealistic UX and User Usability. To back this up, following a tutorial could stimulate more promising results than trying the software with no expert guide. Hence, gathering video content of the participant following the tutorial will help examine the User Usability to see their initial intentions.

Further within the chapter, a few points including:-

"Here's the difference in a nutshell:

- In a **focus group**, a small group of people (usually 5 to 10) sit around a table and talk about things, like their opinions about products, their past experiences with them, or their reactions to new concepts. Focus groups are good for quickly getting a sampling of users' feelings and opinions about things.
- **Usability** tests are about watching one person at a time try to use something (whether it's a Web site, a prototype, or some sketches of a new design) to do typical tasks so you can detect and fix the things that confuse or frustrate them.

" - Steve Krug [16].

Ranging from the suggestion of five to ten respondents, an audience of six surveyees is ample for the survey. Two of these respondents will have done the initial survey to gather data to compare Programming Planner to VisualPro. As the video research is optional, two to three video demonstrations are within the expectations. However, as the survey focuses on the results of VisualPro code generation, both can provide good UX and User Usability results.

'Measuring Usability: Preference vs. Performance', written by Jakob Nielsen and Jonathon Levy [17] experiments to determine whether Preference and Performance influenced each other. The VisualPro results could indicate a clear correlation if the theory stands true. Unfortunately, the authors could not conclude whether they did or not. This analysis aims to determine methods to create a Visual Scripting language that helps novices' with little-to-none experience whilst increasing the overall performance of computing projects. Video content of users' using the VisualPro software provides insight into how the user navigates the software and whether they try any features that are non-existent that show preferences that could lead to feature ideas. A Fuzzy Expert Logic system can help decipher whether the user was using the software effectively.

Fuzzy Expert logic will help identify the performance level to a User Usability category, 'Hard Usability', 'Moderate Usability' and 'Easy Usability'. It is time to use the Fuzzy Expert logic to convert the preference from the initial survey into these categories. The following information could help indicate whether performance and preference contradict each other and may help determine if the initial study did help create a better UX and User Usability result.

Performance Fuzzy Logic Setup:-

Mathematical Key:

$\mathcal{U}$ = Usability Categories: 'Hard Usability', 'Moderate Usability' and 'Easy Usability'.

$\mathcal{P}$ = Performance Categories: 'Very Low', 'Low', 'High' and 'Very High'.

$\mathcal{VL}$ = Performance Category: 'Very Low (0%)'. - $\mathcal{L}$ = 'Low (25%).'

$\mathcal{H}$ = Performance Category: 'High (75%)'. - $\mathcal{VH}$ = 'Very High (100%)'.

$\mathcal{R}$ = Responses with Performance Categories. - $\mathcal{T}$ = An array of responses.

$$\mathcal{U} = \sum_{n=0}^{3} \frac{100}{3} - (n = n + 5) \rightarrow \left| \begin{array}{cccc} \mathcal{HU} & \mathcal{MU} & \mathcal{EU} \end{array} \right|$$

$$\mathcal{P} = \frac{100}{4} \rightarrow \left| \begin{array}{cccc} \mathcal{VL} & \mathcal{L} & \mathcal{H} & \mathcal{VH} \end{array} \right|$$

$$\mathcal{R}_P = \left| \begin{array}{c} T_n \end{array} \right|$$

$$\sum_{n=1}^{n<R_n} \mathcal{M}_{\setminus} = \frac{R_i}{R_T}$$

Essentially to work out the Performance figures, calculation of the code submissions within the survey to generate a percentage. Four statements measured by $\mathcal{P}$, 'The respondent followed the task correctly', 'The respondent seems to have followed the task correctly', 'The respondent seems to have followed the task incorrectly' and 'The respondent followed the task incorrectly' helps to determine the performance level. Each respondent's results are tallied up and positioned in the $\mathcal{R}_{\mathcal{P}}$ array, which provides a performance level of each respondent. Sum of $\mathcal{P}$ of each $\mathcal{R}$ element and then divided by the $\mathcal{T}_{\setminus}$ provides an average percentage. The $\mathcal{U}$ identifies the Fuzzy Expert categories percentage with added weights. The weighing strategy should enable the analysis to be unbiased.

From the initial survey, a selection of two preference questions including Figure 5.19 - 'According to you, how should the features of the Graphical User Interface have to work?' and Figure 5.20 - 'How would you prefer to access the software?' helps work out the preference level. A calculation to sort out each categories with the responses and whether or not the preference is within the VisualPro design implementation. To filter the preference down, only the preferences within the VisualPro design implementation are taken into account. The mean of the preference of each category is then calculated and then inserted into the Fuzzy Expert logic. The preference and performance levels are then compared to determine the overall User Usability and analyse whether preference affects performance.

4. PROTOTYPE

To access the prototype that demonstrates communication from a .NET Framework 4.7.2 to a C++ backend library, go to Prototypes/GUI Prototype .NET-C++ [18]. Even though this prototype is not related to the current project, it demonstrates how the frontend can interact with the backend and provides insight into how this is possible. The purpose of the application is not to measure the application's performance merely to provide insight and experience on how the application communicates to the backend.

The source code, found in Prototypes/GUI Prototype .NET-C++ [18], demonstrates the functionality to load the DLLs within the application. Implementation of two techniques follows:-

- *DllImportAttribute* (Enables the ability to import Native DLLs into the application).
- *_declspec(dllexport)* (C communication files use this to create entry points.)

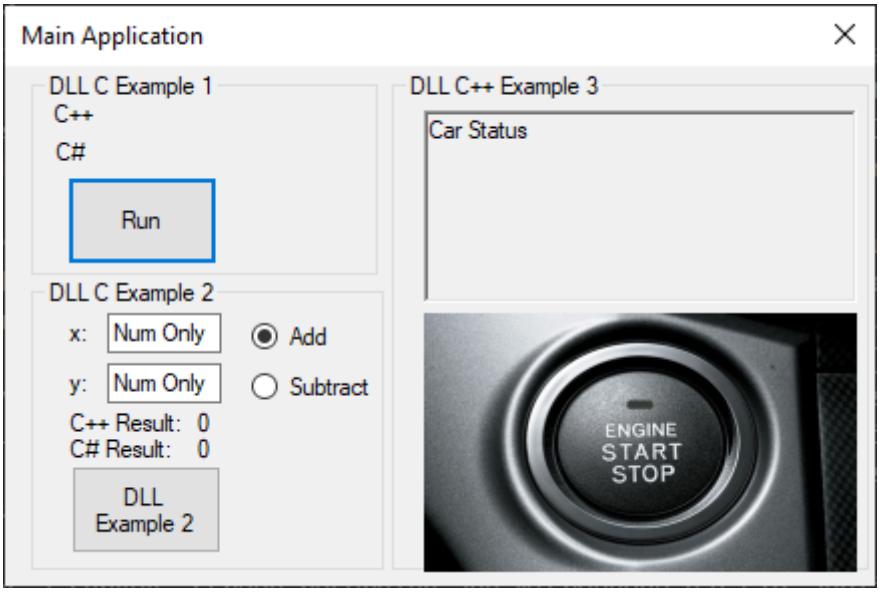

Fig. 4.1. GUI Prototype Communication C# to C++

5. DESIGN AND FEEDBACK

The Design and Feedback section covers the Research Methodology and the Pre-Planning, which shows the steps and the level of thought that went into deciding VisualPro's purpose and the existing software and prototypes. Some Notebook Planning may not make sense as some foreseeable problems were thought and planned ahead of time.

5.1. Pre-Planning

5.1.1. HTML Design

Figure 5.1 shows a HyperText Markup Language (HTML) Design of how the UI should look. The Empty Column is where the user drops elements like Classes, Functions or Variables. Buttons on the right can be dragged and dropped on the Empty Column to the left. The functionality of the drop should create a new container, giving the user acknowledgement there's a new parent or sub-element with room for configuration and another drag and drop container within the class or function.

Fig. 5.1. HTML Design - Found at: Original Image

5.1.2. Notebook Page 1-2

Figure 5.2 contains numerous ideas. Page 1 contains a heading, 'Personal Assistant', which indicates the thought process of a personal assistant before the Visual Programming Scripting program progressed. After planning this, it became apparent to use a type of Artificial Intelligence (AI), which knew the basic syntax of different languages and would write or advise the user when typing. This process is very similar to TabNine [19].

The second page of this figure, shows a few different examples of IF statement syntax from lafnguages to find a 'dynamic' common rather than keep it static like the previous software. The sketch below shows a name for the software and a example of how the language selection would look on UI.

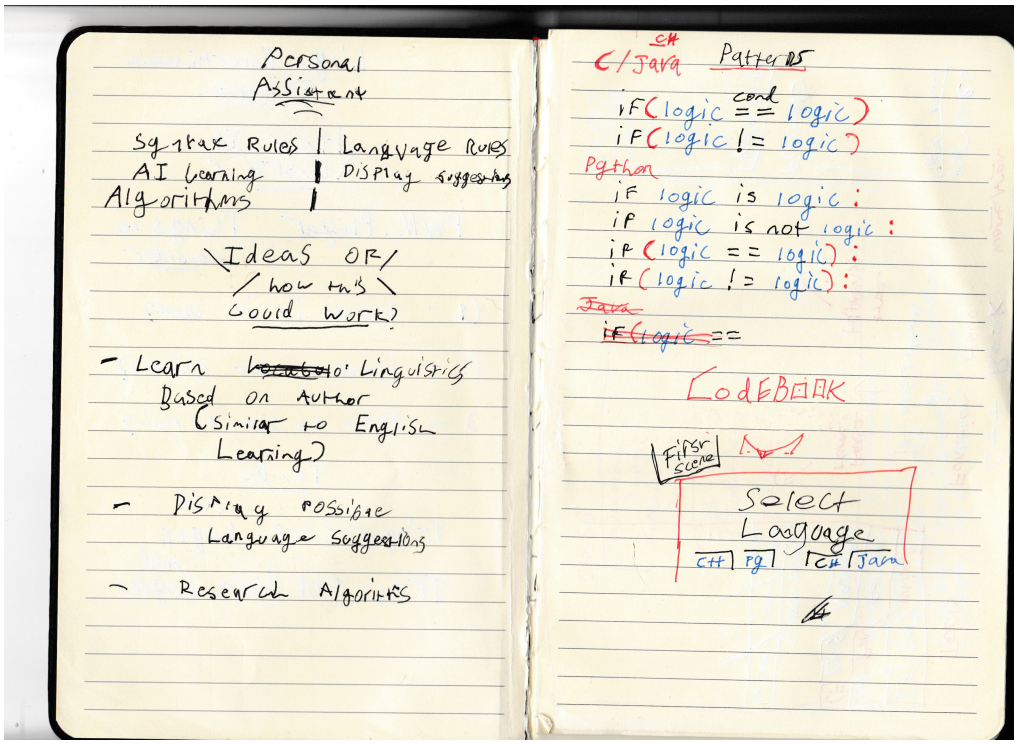

Fig. 5.2. Notebook Plan Page 1-2 - Found at: Original Image

5.1.3. Notebook Page 2-3

Figure 5.3 displays two pages of how the Visual Scripting could look. The top page shows the tools on the left, such as Logical, Structure and Scoping tools. On the right, it should show a list of current variables, functions and classes in a hierarchy style. The middle of the application is the work area.

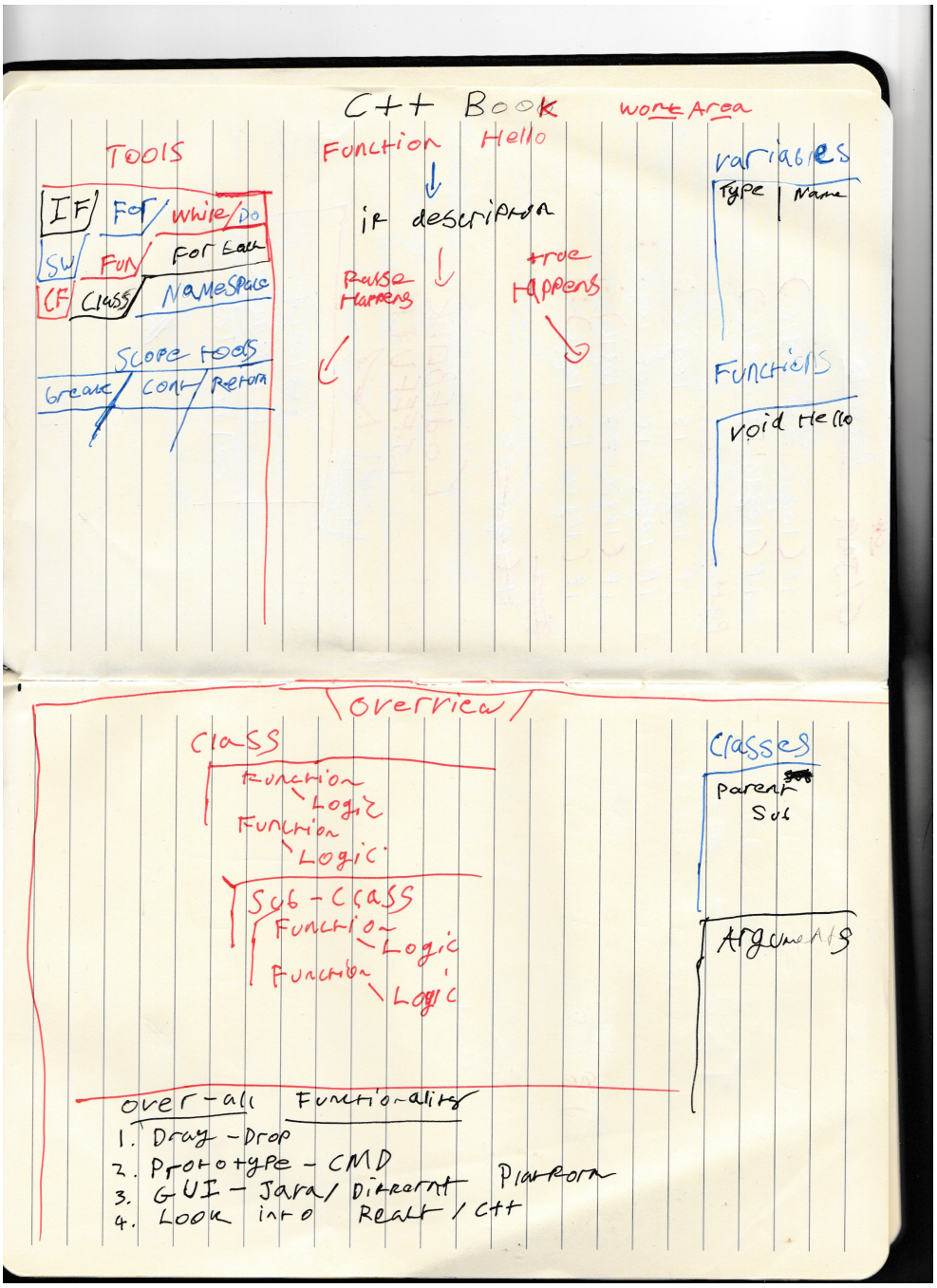

Fig. 5.3. Notebook Plan Page 3-4 - Found at: Original Image

5.1.4. Notebook Page 5-6

Figure 5.4 gives the planning of how the syntax for most programming and scripting languages. The plan points out patterns and tries to identify their names for the Extensible Markup Language (XML) and backend to comprehend. This piece of planning helped excel the development of the program's dynamic side to allow the addition of new languages to the software.

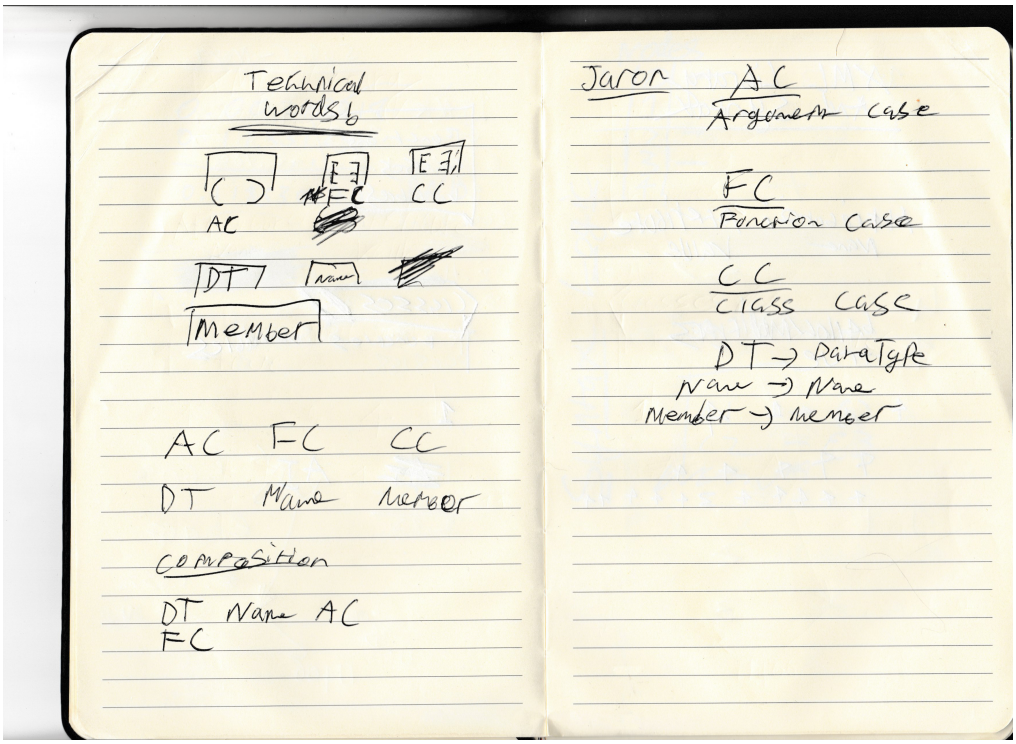

Fig. 5.4. Notebook Plan Page 5-6 - Found at: Original Image

5.1.5. Notebook Page 7-8

Figure 5.5 demonstrates how the XML would possibly look. The table shows N (Node), A (Attribute) and the Meaning. To explain what is going on, Node |0| is the header node (holds library data such as **system** or **iostream**), holds three attributes. Attribute |0| holds 'Type', which is the purpose, Attribute |1| holds the 'Name', which is the library's name. Attribute |2| holds the 'Syntax Value', which in C# it is *using ;* and in C++, the value is *#include <>* .

According to the second node |1| holds information for structure or classes in languages. This node contains three attributes. Attribute |0| holds 'Name of Struct *(or)* Class', which contains the user-defined name they have created inside VisualPro. Attribute |2| holds the 'Properties' like the member of the struct/class. Attribute |3| holds the syntax value, the open and close case of a struct/class and layout within the struct/class.

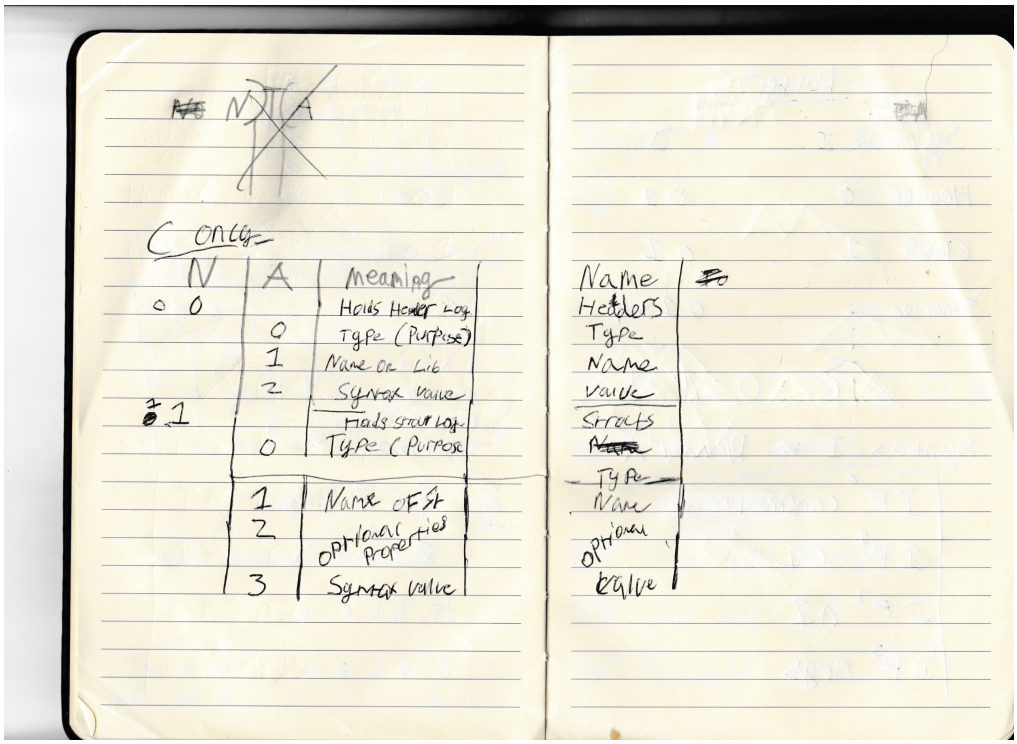

Fig. 5.5. Notebook Plan Page 7-8 - Found at: Original Image

5.1.6. Notebook Page 9-10

Figure 5.6 answers how the loops and logical statements work in the LanguageCompiler library. This planning shows how the XML nodes and the Planner List should work together. In theory, the Planner List, of what the user populates with the program's use, combines with the XML document with the chosen language.

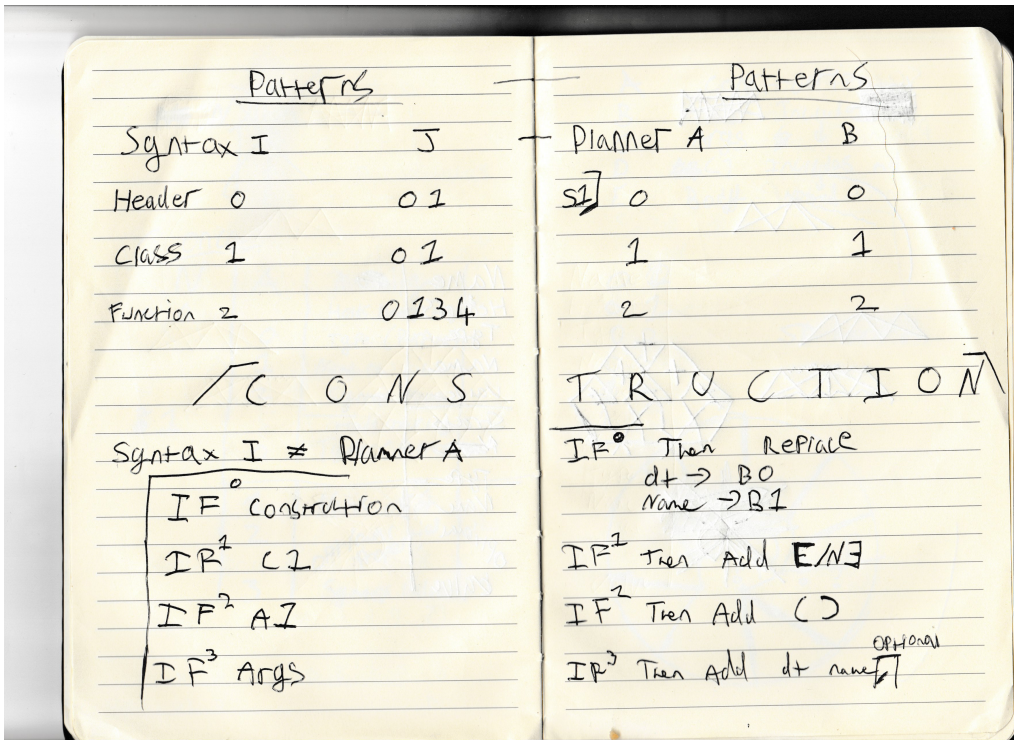

Fig. 5.6. Notebook Plan Page 9-10 - Found at: Original Image

5.1.7. Notebook Page 11

Figure 5.7 displays how the function node should work. This miniature theory is to help the developer understand how the XML document should work in a progr ammatical sense. This again helped when it comes to the prototypes further in this document.

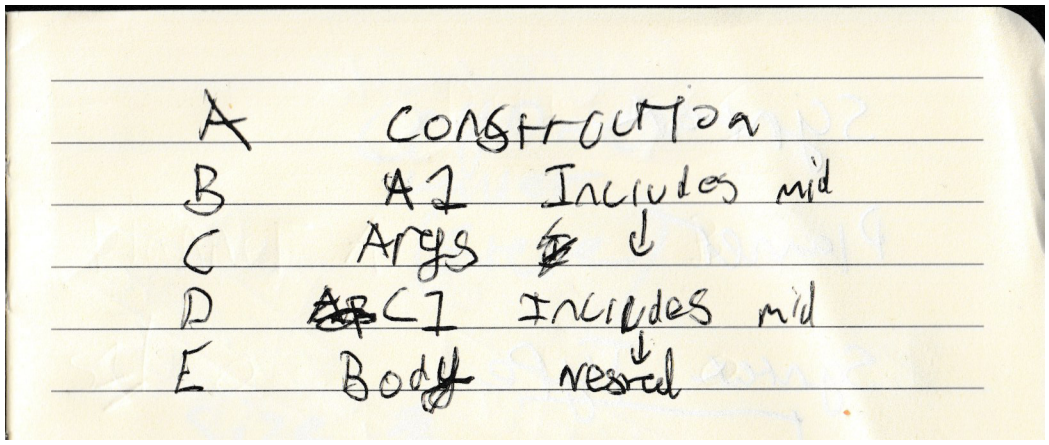

Fig. 5.7. Notebook Plan Page 11 - Found at: Original Image

5.1.8. Notebook Page 12-13

Figure 5.8 shows a predicted problem is as follows:

Description:

How will the program know where to put a sub-child and how will it?

Answer:

The planner would hold its parent and sub identity, and the Triangle (30 Code) symbol tells the software where to put the child. The Triangle symbol will only appear of classes or functions that have children. This means that the program will focus on parents, close the tags, then rescan to put the children in the code. The problem before is that it would be hard to tell the program to remember when and where to close tags if it got too deep in theory.

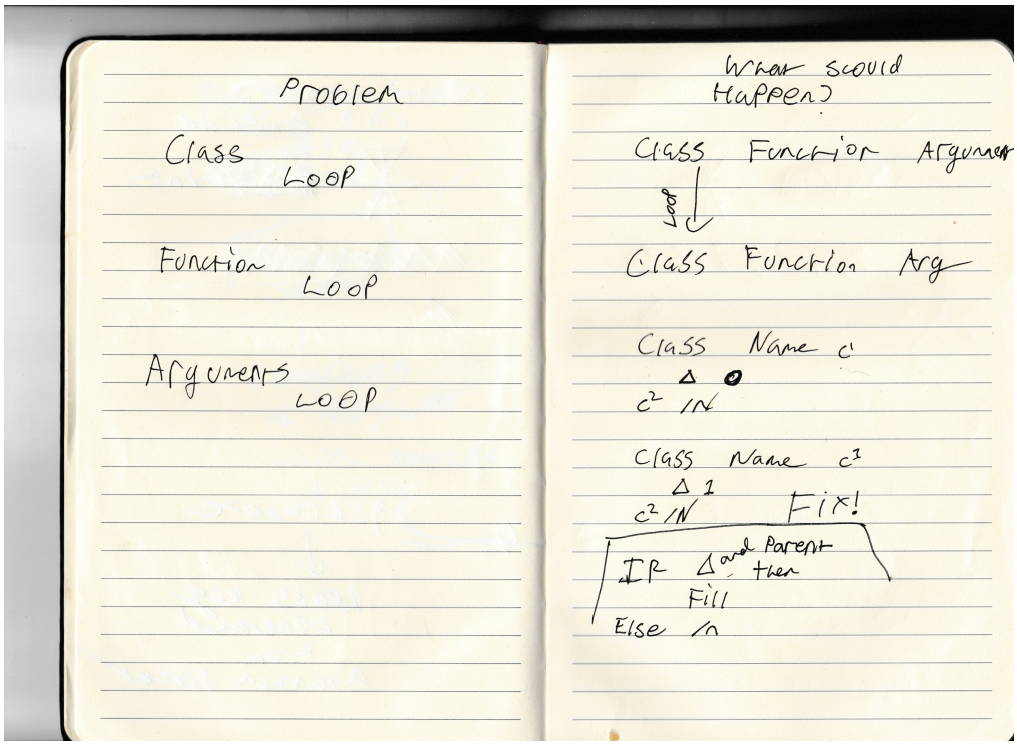

Fig. 5.8. Notebook Plan Page 12-13 - Found at: Original Image

5.2. Results of Feedback

5.2.1. Survey Question 1

According to the chart 5.9, *87.5%* chose 'Programming Planner Improved', and *12.5%* chose 'None of the Above' to the question, 'As a developer, which program worked for you?'. This data suggests that most of the responses preferred 'Programming Planner Improved' out of eight. The response may suggest that participants enjoyed the interface as it offers more than the Programming Planner. However, the *12.5%* of the 'None of the Above' option displays that the application may prove tedious for some.

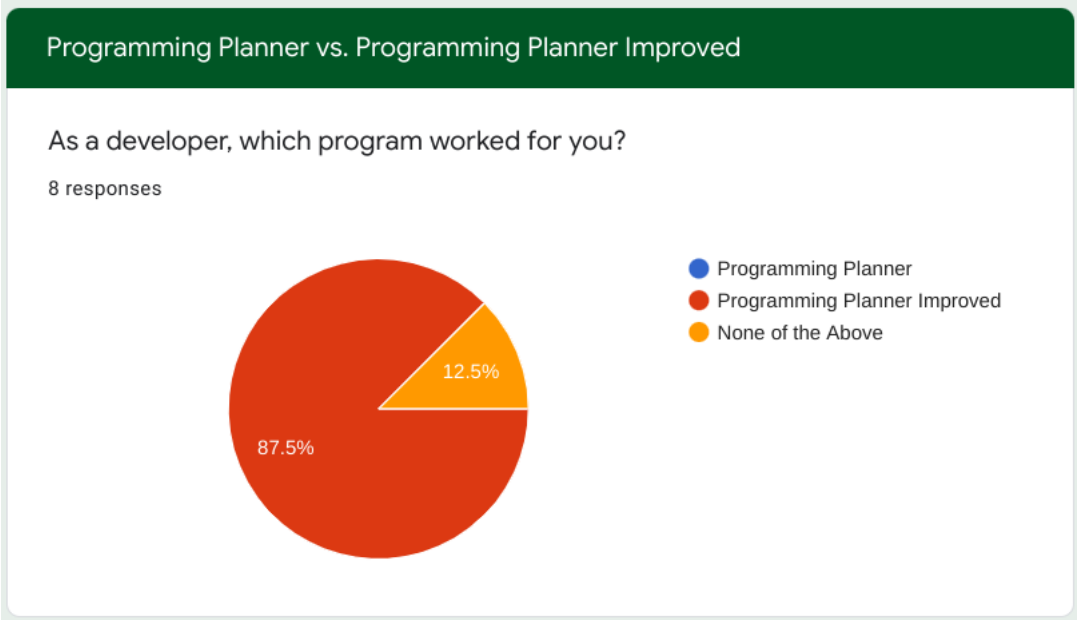

Fig. 5.9. Survey Question 1 - Found at: Original Image

5.2.2. Survey Question 2

Figure 5.10 asks the question, 'If the last option on the previous question is ticked, then explain why?'. Four responses replied with:-

- 'More flexibility' -
  This response could mean two things: the Programming Planner Improved is more flexible than the first program or that both programs would benefit from more flexibility.
- 'Both work' -
  This response indicates that both software works.
- 'The option to choose the desired programming language at the end of the programming' - A surveyee liked the option in Programming Planner Improved to select different programming languages.

- ‘Because it had more options and had example code at the end of planning’ - This answer describes that the Programming Planner Improved offered more options, such as argument selection and code language.

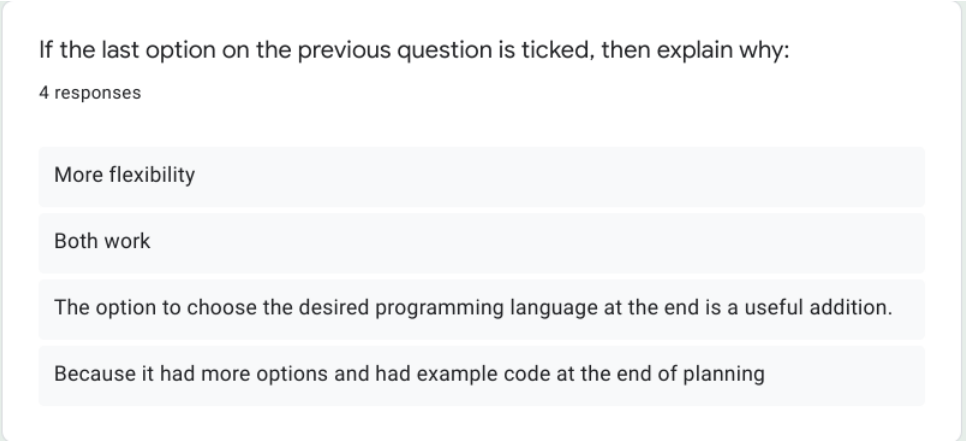

Fig. 5.10. Survey Question 2 - Found at: Original Image

5.2.3. Survey Question 3

Figure 5.11 asks the question to Surveyee’s, ‘Did the tutorial at the beginning of Programming Planner Improved help navigate around the application?’. This question tries to find out if the application is hard to use overall. The responses were 87.5% for yes, and 12.5% for no. Even though the most of the responses chose yes, the application may still be hard to use if, theoretically, one out of eight users found the UX frustrating or confusing.

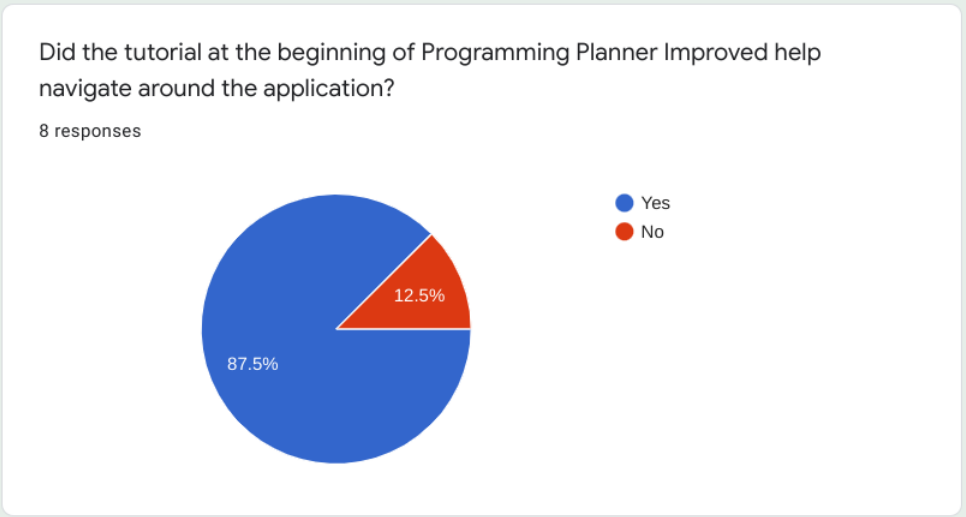

Fig. 5.11. Survey Question 3 - Found at: Original Image

5.2.4. Survey Question 4

Figure 5.12, 'What makes the Programming Planner Improved better than the previous product?' checkbox question, offered five qualities. These are:-

- **Ease of Use** - Aimed to find out how easy the application is easy to use. 50% of the responses think it was easier to use than the previous product. This feedback could mean to things, the first product was similar and consequently did not improve the application in this quality or that the product needs to improve in this area.
- **Performance** - To see if the audience could notice any performance difference. This quality is vital as a Graphical User Interface (GUI) can somewhat be 'chunky'. The responses were 0% out of the responses; though this may seem negative, it is sometimes hard to notice the difference between console and desktop applications.
- **Extra Features** - The quality, 'Extra Features', displays if the extra features in the new application stood out and improved the previous application. The responses were 62.5%, which indicates that the extra features did stand out and made the application better.
- **File Structure** - The quality, 'File Structure' of the second application is revised and stores the saved file in a directory other than the ROOT directory. 25% of the responses chose this option, which means that the File Structure did not impact the application as effectively as the 'Extra Features' quality.

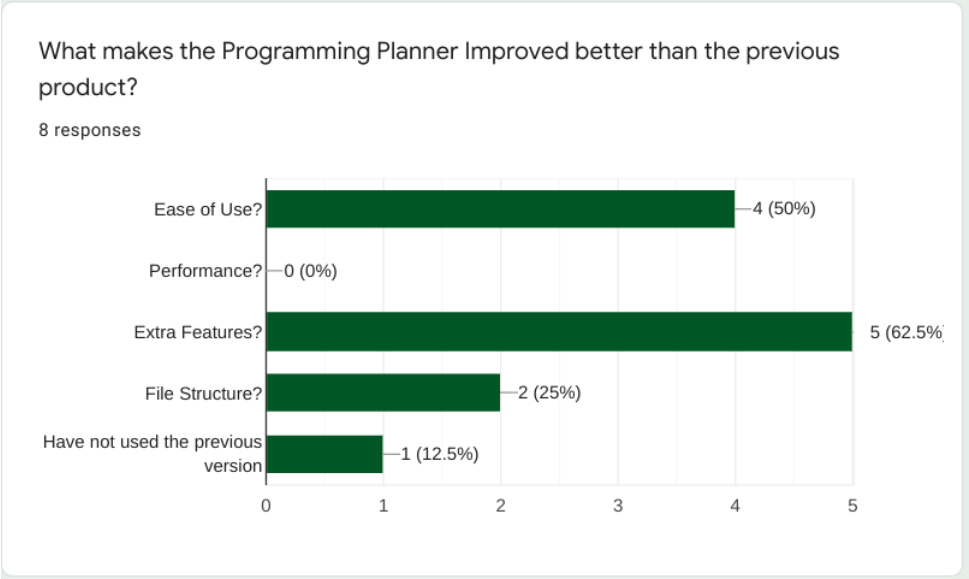

Fig. 5.12. Survey Question 4 - Found at: Original Image

5.2.5. Survey Question 5

Figure 5.13 asks the Surveyee's, 'Explain your answer' to the last question. This question will tell what the Surveyee's felt when answering the last question. The responses are as follows:

- 'More features allowed for a wider range of approaches' - This specific feedback suggests that it is better to add more functionality in the GUI application to allow for more usability.
- 'More languages and was easier to use. Very impressed' - This response implies that the idea of a dynamic language interface to allow users to add more languages will increase User Experience.
- 'Easier to use, the better' - Visual Scripting already is complicated when looking at the Unreal Engine's Visual Scripting interface. If it is possible to create a lightweight Visual Scripting pad to allow a programmer to create code snippets, it could bring more developers into the field and increase production time already in the field.
- 'Have not used the previous version' - This suggests that the Surveyee only tested Programming Planner Improved.
- 'Allowed me to create a function structure and save it. Seemed difficult to follow (variables weren' numbered)' - According to this response, the user found it difficult to follow through and should be something to think about when making the GUI application.
- 'Easy, helpful and understandable' - This response shows that the user found the software easy to use, helpful and understanding during their experience.
- 'The extra features added to the improved version of the program improve used experience by adding expanded usability' - The response suggests that again the extra functionality helped improve the User Experience, in Programmer Planner Improved.
- 'It just had extra features and seemed more useful' - Which implies that the user found the extra features more comfortable and useful to use.

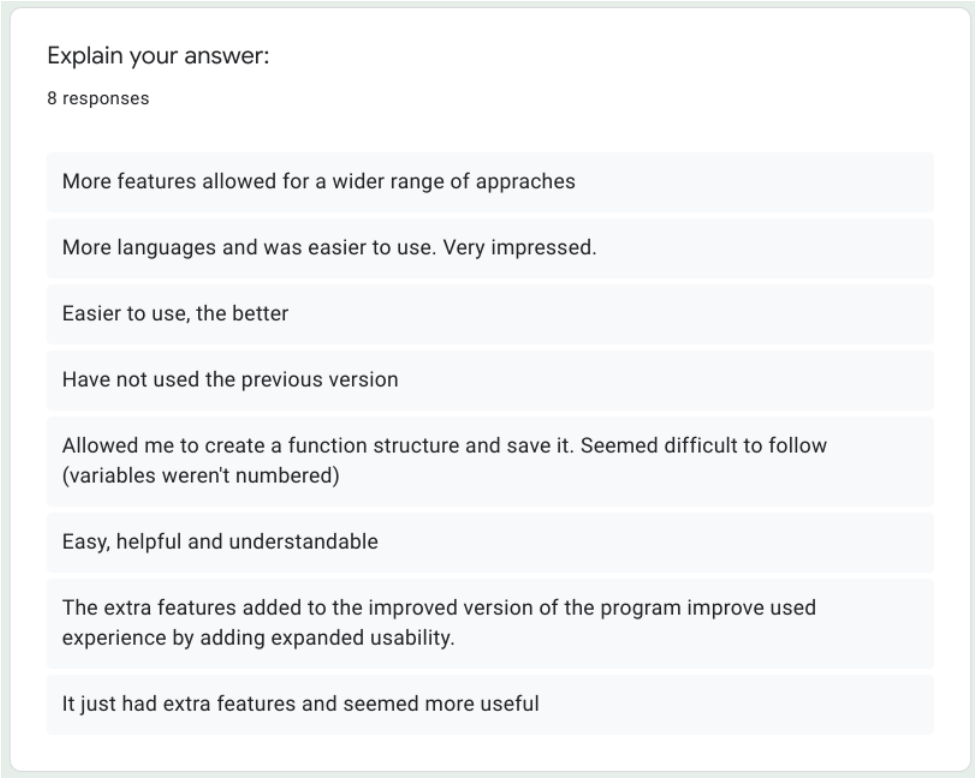

Fig. 5.13. Survey Question 5 - Found at: Original Image

5.2.6. Survey Question 6

Figure 5.14 asks the Surveyee, 'How would you rate the User Experience (overall)?'. The question tries to aim for an idea of how the UX of the two products. On a scale of one (Stressful) to ten (Relaxing), 62.5% of the responses went with option eight, 37.5% of the responses went with option seven. These figures show that the UX has room for improvement. When designing and implementing the GUI product, it is important to think about the UI and UX.

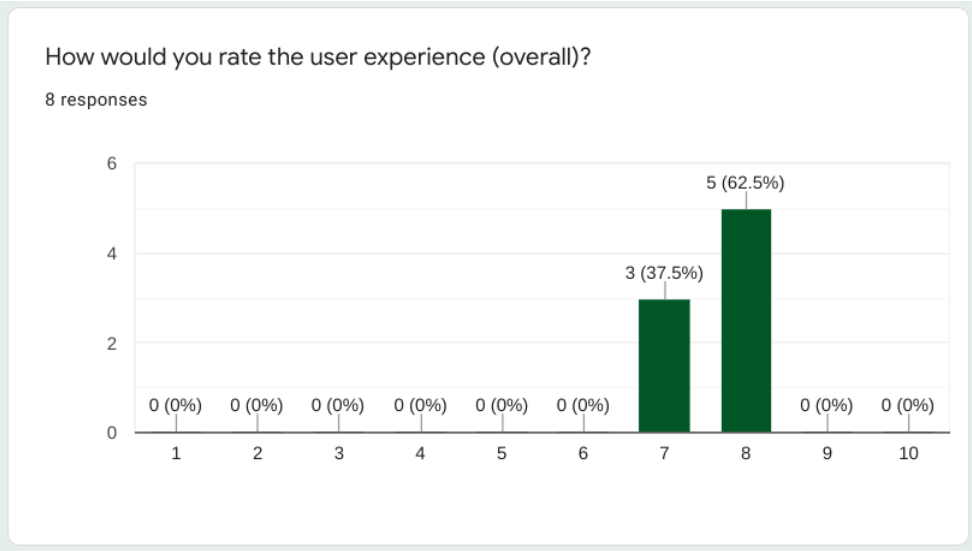

Fig. 5.14. Survey Question 6 - Found at: Original Image

5.2.7. Survey Question 7

Figure 5.15 asks the question, 'How useful is Programming Planner for beginners?'. This question is done on a scale of one (Complicated) to five (Useful). The feedback is ranged from one to five and suggests that beginners may find this platform hard work. The responses are as follows:

- **One** - 25% responses. These respondents reckon that Programming Planner is complicated for beginners.
- **Three** - 12.5% responses. These respondents thought that Programming Planner is a little complicated for beginners.
- **Five** - 50% responses. These respondents figure that Programming Planner is close to Useful for beginners.
- **Six** - 12.5% responses. These respondents consider that Programming Planner is Useful for beginners.

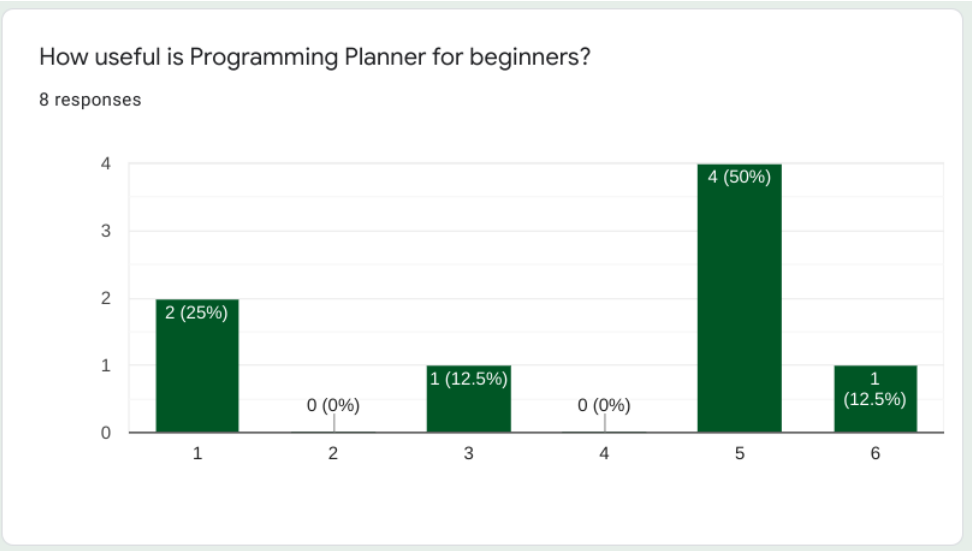

Fig. 5.15. Survey Question 7 - Found at: Original Image

5.2.8. Survey Question 8

Question 5.16 ‘Did this help create a structure of your favourite language faster than standard methods?’. 75% of responses said Yes, 12.5% of responses said ‘Would not recommend to new programmer as they will not learn the basics’ and 12.5% said ‘I have not used other methods’. This chart suggests that the majority found this software to be faster than their standard methods.

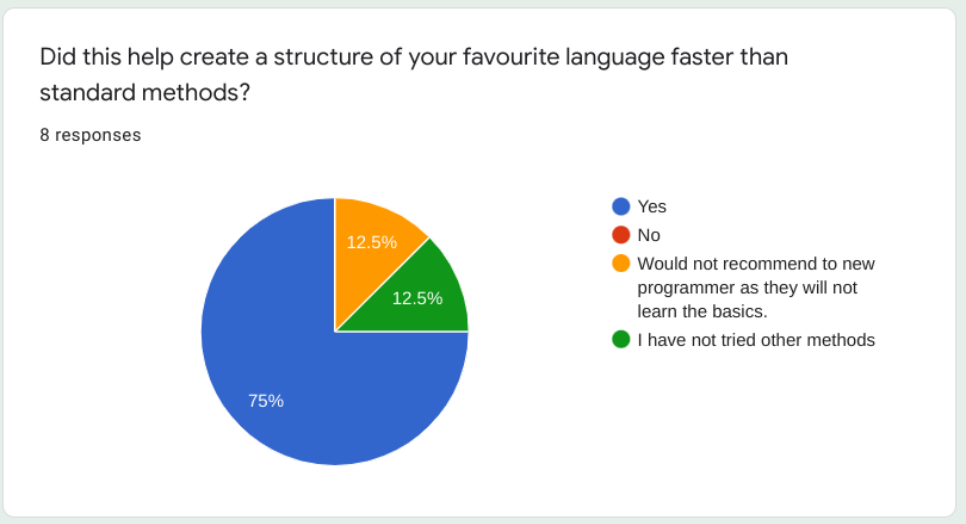

Fig. 5.16. Survey Question 8 - Found at: Original Image

5.2.9. Survey Question 9

Question 5.17 ‘What features would you wish Programming Planner to offer?’. This question attempts to determine if the user has any requests about the software they have

used. These were the responses:-

- ‘When I wanted only one function on the variable creation it asked if it works with other functions but I didn’t have any other functions’ - This suggestion shows that the user was confused with the way the program asked the user if the variable was a argument or a standard variable.
- ‘The target audience for this program will have little coding experience, and may forget what is required of a function, variable, etc. Adding examples to each page may be a good idea to reduce confusing.’ - This suggestion shows that a help page or tips pop up on creating an element within the GUI program will help beginners use this software.
- ‘A set of pre made open source programs that could be quickly accessed’ - Perhaps, the program could offer a GitHub repository import, which will examine the code and visualise it for the user.
- ‘Would be better with visualisation if writing a longer program’ - The response could mean two things. It could mean that the user wanted to have an ability to overview the code within the console application, or it could mean that the user would have preferred a GUI work area.
- ‘A GUI’ - This response means that the user would have preferred a GUI product rather than a console application.
- ‘Allow sudo code comments to be added to the function’ - In the GUI application, it could have the ability for the user to enter comments.
- ‘Calculator?’ - This response did not seem useful to add in to the future software, as popup calculators are available in most operating systems.

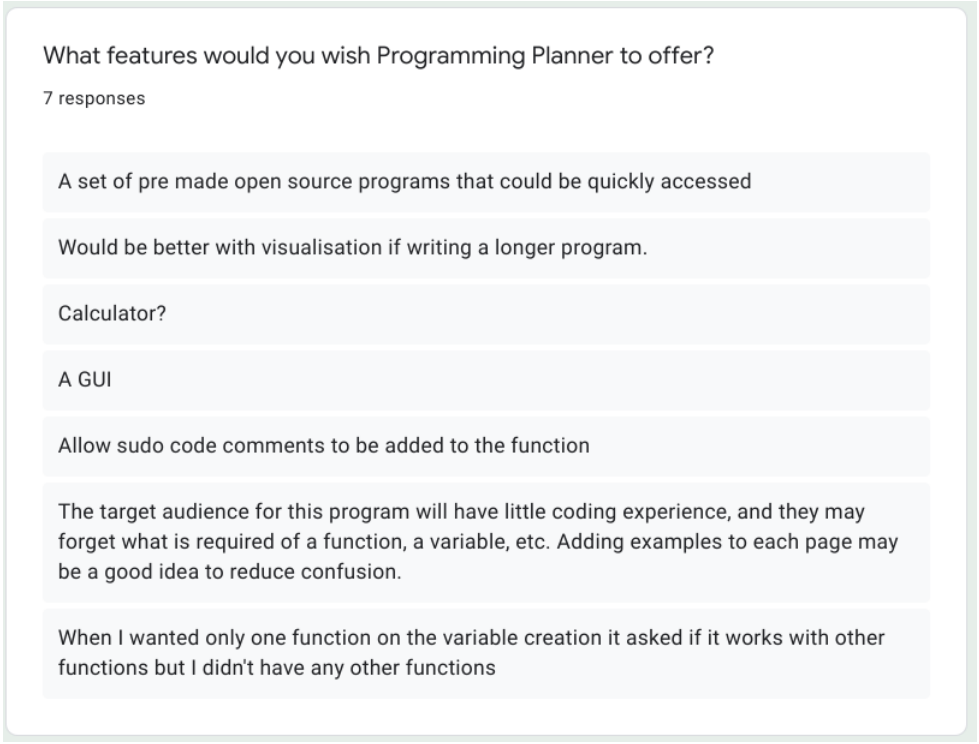

Fig. 5.17. Survey Question 9 - Found at: Original Image

5.2.10. Survey Question 10

Figure 5.18 ask 'Would a Graphical User Interface uplift the User Experience?' as a multiple-choice question. The 100% chose the yes choice. This feedback implies that they would like a GUI to work with over a console application out of the eight responses.

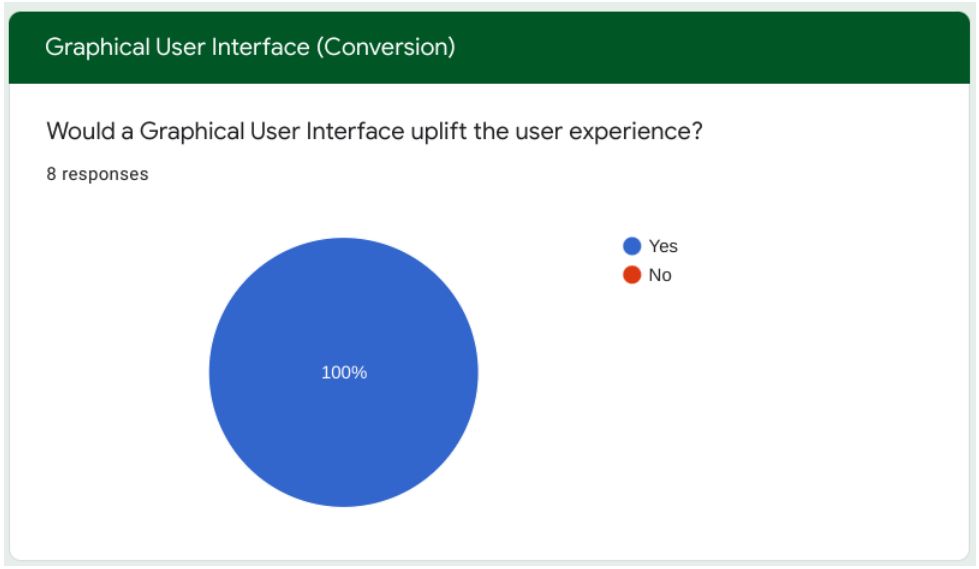

Fig. 5.18. Survey Question 10 - Found at: Original Image

5.2.11. Survey Question 11

A grid question asked ‘According to you, how should the features of the Graphical User Interface have to work?’. The following options are: ‘Clickable’, ‘Drag and Drop’, ‘Drop Down Menu’ and ‘Text Filled’. The details are ‘Structure Elements’, ‘Logic Elements’, ‘Select Languages and Save’ and ‘Sub Elements’. After overlooking the table 5.19 ‘Structure Elements’ looks like it favoured Drag and Drop the most by four responses. The Drag and Drop were also voted for the ‘Logic Elements’ by five responses. The ‘Select Languages and Save’ preferred the Drop Down Menu by six responses, and lastly, the ‘Sub Elements’ liked the idea if it was Text Filled.

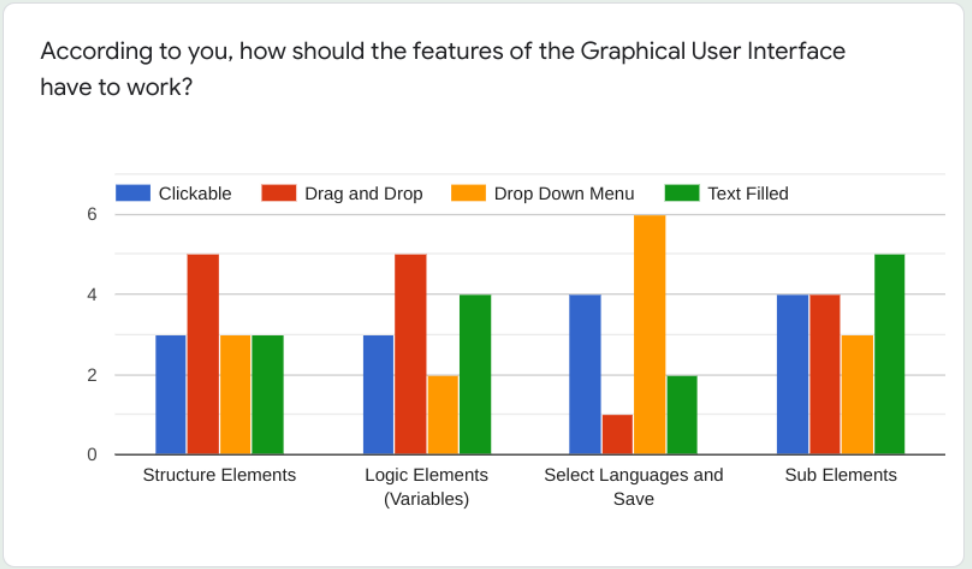

Fig. 5.19. Survey Question 11 - Found at: Original Image

5.2.12. Survey Question 12

This grid asks the participants, ‘How would you prefer to access the software?’. The options are ‘Windows Operating System’, ‘Linux Operating System’, ‘ChromeOS/Android’ and ‘iOS’. The details are ‘Computer, Laptop or Tablet Devices’, ‘Web Application’, ‘Mobile Devices’. For the first column, the responses favour the Windows Operating System by eight. The second column, Windows Operating System, is also preferred by eight. The last column, ChromeOS, is the six responses favourites, and iOS is preferred by four. After reviewing this, the application aims for the Windows Operating System to get the most attention. Programming Planner and Programming Planner Improved works for both Windows and Linux Operating Systems.

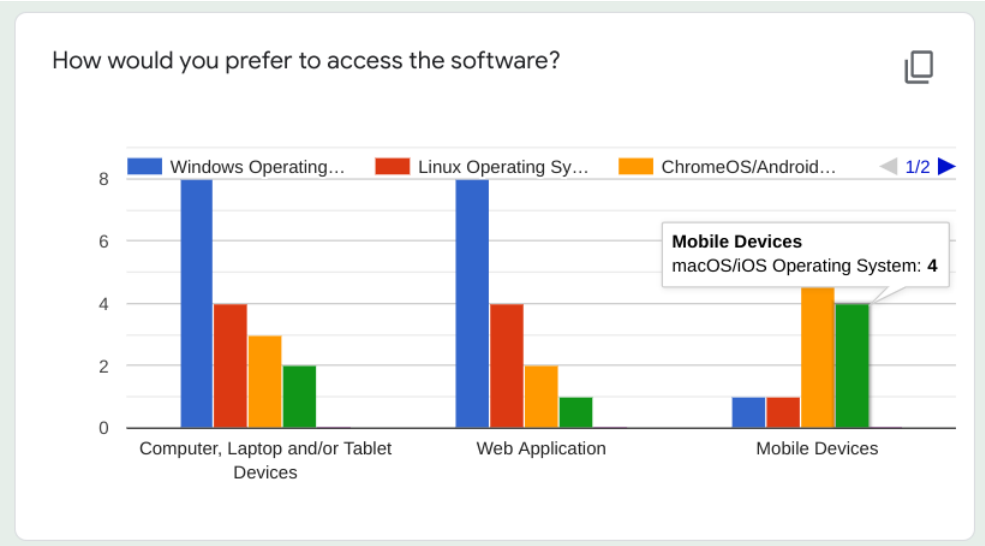

Fig. 5.20. Survey Question 12 - Found at: Original Image

5.2.13. Survey Question 13

Figure 5.21 a question, 'Any comment?' follows the last questions. The only response was 'No'.

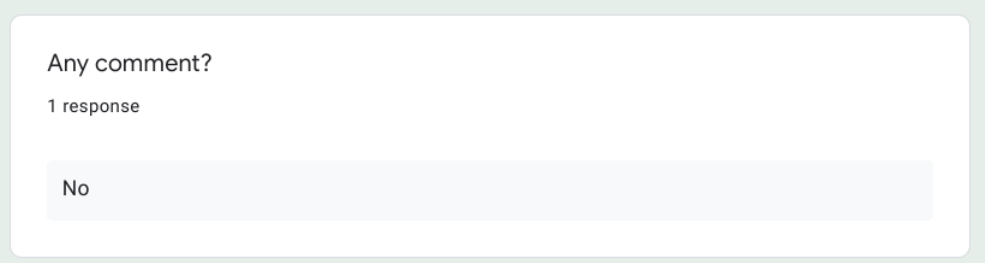

Fig. 5.21. Survey Question 13 - Found at: Original Image

5.2.14. Survey Question 14-15

Figures 5.22 and 5.23 shows two questions, 'How complicated is Visual Scripting? (Reference to Unreal Engine for an example).' Furthermore, 'How complicated is Website Builders?' these questions portray how existing Visual Scripting Web Builders are successful, and if it is possible to do a service that is easier to use, similar to a web builder. The question is on a scale of one (Piece of Cake) to five (Complicated). The feedback from the first question:-

- Three - 37.5% of the responses. This indicates that this percentage of the participants find Visual Scripting moderately easy.
- Four - 50% of the responses. This displays that 50% of the participants find Visual Scripting a bit more complicated than the 37.5%.

- Five - 12.5% of the responses find Visual Scripting tough.

The feedback from the second question:-

- One - 25% of the responses find Web Builders a ‘Piece of Cake’.
- Two - 50% of the responses find Web Builders more or less easy.
- Three - 25% of the responses find Web Builders moderately easy.

These findings will change the research perspective to look more into the Web Builder layouts, functionality and UX to work out the GUI of the product.

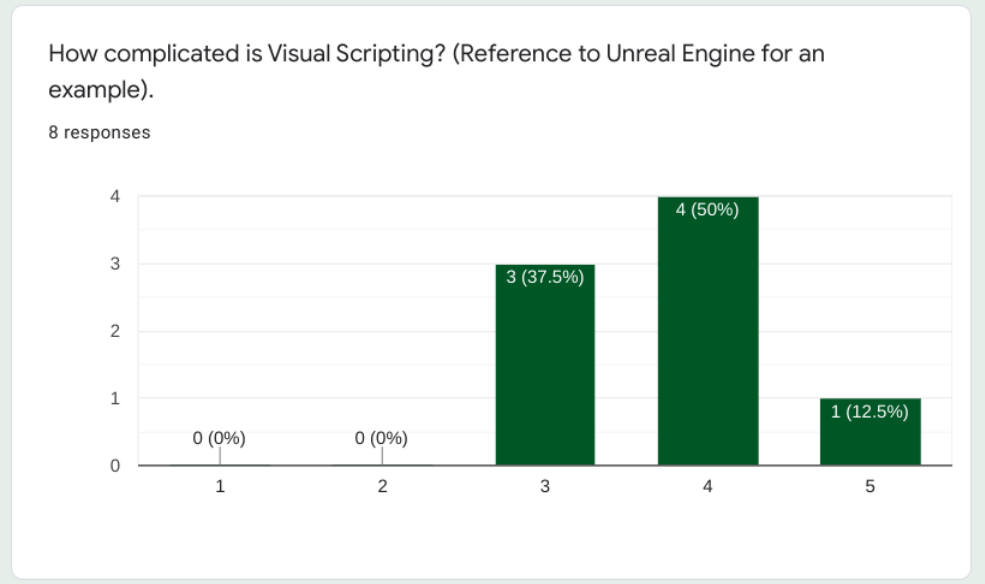

Fig. 5.22. Survey Question 14 - Found at: Original Image

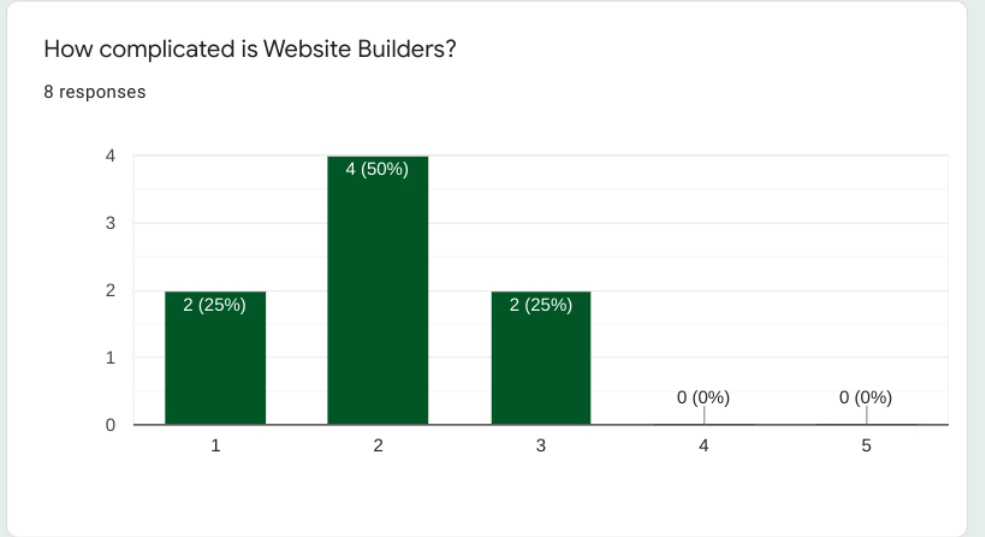

Fig. 5.23. Survey Question 15 - Found at: Original Image

5.2.15. Survey Question 16

Question 5.24, 'If Programming Planner had an easy-to-use drag and drop feature without input and output relationships, then will this improve Visual Scripting overall?', the question paints a picture from the participants to see if the relationships in Visual Scripting that is similar to Entity-Relationship Diagrams if they cause a problem. All eight responses replied back with yes.

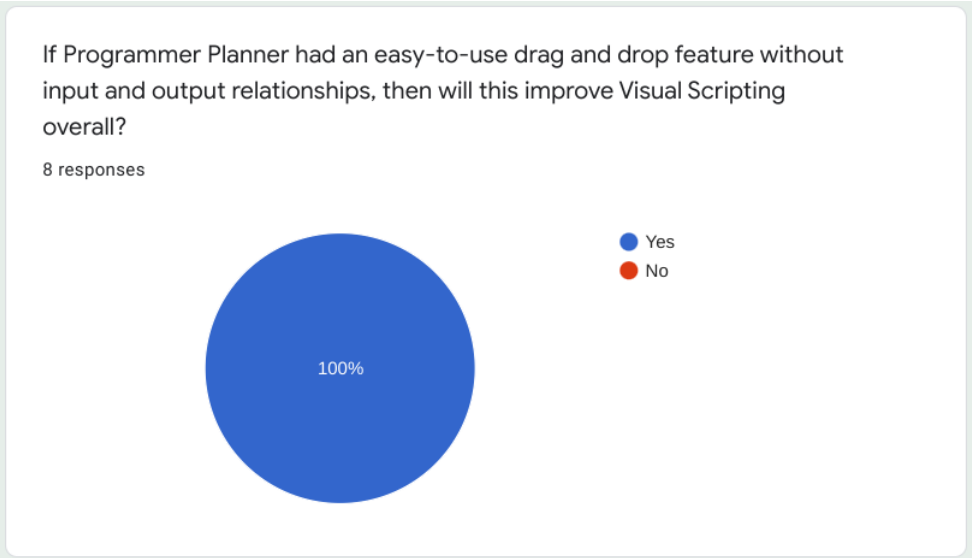

Fig. 5.24. Survey Question 16 - Found at: Original Image

5.2.16. Survey Question 17

Similar to the previous question to paint an image from what the participants want to see, 'Would Visual Scripting be better if it allowed users to add any language they wish and generate any language with one click of a button?'. According to the figure 5.25, all of the eight responses responded with yes.

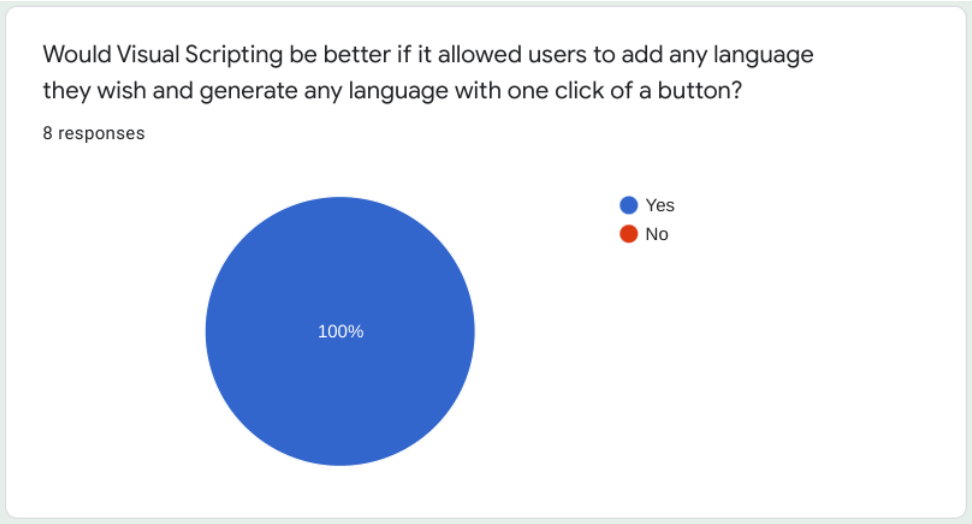

Fig. 5.25. Survey Question 17 - Found at: Original Image

5.2.17. Survey Question 18

To squeeze more information from the participants, a question, 'What was the reason for your selection?' was asked and made compulsory. The responses are as follows:-

- 'Would be easier to write programs (for some people)'
- 'A feature such as this would greatly reduce the production times involved in program development.'
- 'save time and allow users to focus on the details'
- 'I would like to code in any language I choose if that option was applicable.'
- 'Useful feature'
- 'Makes it more visual'
- 'Again, the easier to use, and the more features, the better'
- 'More accessibility'

After reading the feedback, a few points are to address. The current software already makes it easier to write code for some people, so it is imperative that the new software does not distract but improve the capability of writing software. The command-line may benefit users as it keeps developers in a similar environment they are probably already using. Another point to address is this question, 'Will the GUI software get too heavy with too many features making it no longer lightweight?'. The best thing about having Dynamic-Link Libraries is that developers can incorporate into Java Android, iOS, Web Applications and, but not limited to, Visual Studio Code Extensions. The backend can

open a ‘can of worms’ to allow different interfaces to the top of the backend, like digital notebooks or other ideas. Anyone who develops on top of the backend can make any interfaces or even new scripting languages. This idea, in turn, will enable others to make it easier to use Visual Scripting and Languages to Write. For example, Python or JavaScript has the potential of being converted into C++ and vice-versa.

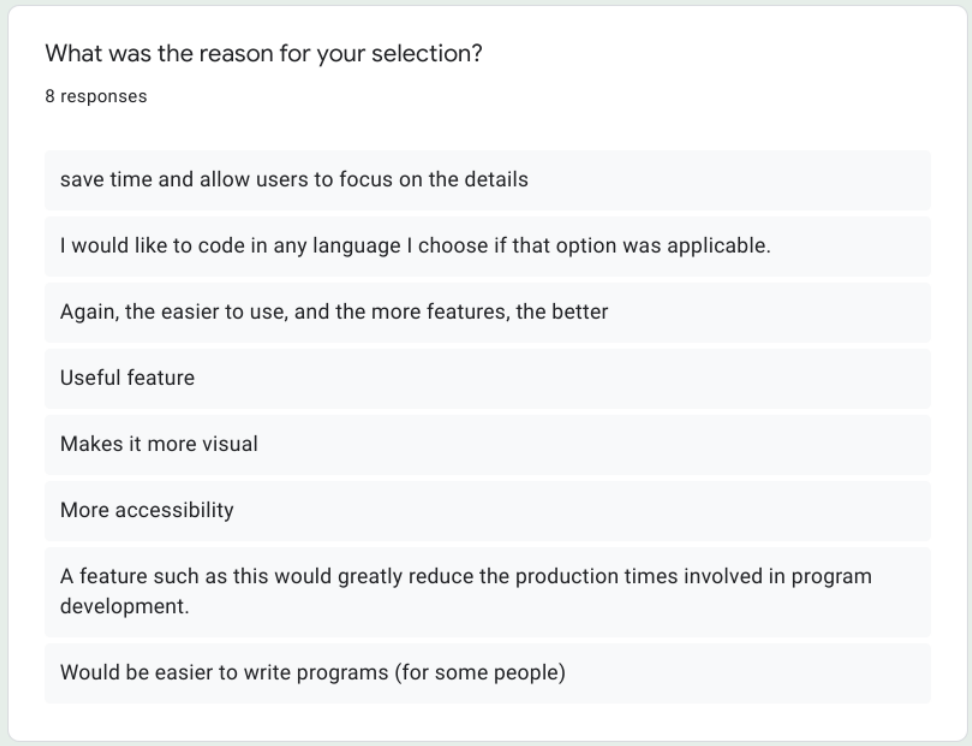

Fig. 5.26. Survey Question 18 - Found at: Original Image

5.2.18. Survey Question 19

Final question which asked the participant, ‘Would you like to leave any extra feedback?’, which hopefully gets any extra feedback that the survey missed out. Six responses, excluding two including the response ‘No’, are displayed below:-

- ‘Introducing both drag and drop, as well as text filled features will appeal to both beginners and experienced programmers.’ - The reply suggests that both Drag and Drop menus and Text Filled features will appeal to both programmers than just one of the methods used. Another idea this program opens up is creating snippets that can work in Visual Studio Extension, giving the user more flexible tools for development.
- ‘A training manaul along with some begginer tutorials would really help new users get to grips with the software’ - A training manual or documentation would benefit both the console and GUI applications.

- ‘Command-line is hard on eyes when using. Not very good for Visual Scripting in general.’ - The best thing about the console application is that it already has a language compiler, which plays a massive part to give the GUI application a backend. However, this response backs up the reasoning why the frontend matters in this project.
- ‘Nothing besides a nice GUI’ - Many Visual Scripting is done in GUI applications, so it was unusual for a console application to have achieved a similar result.

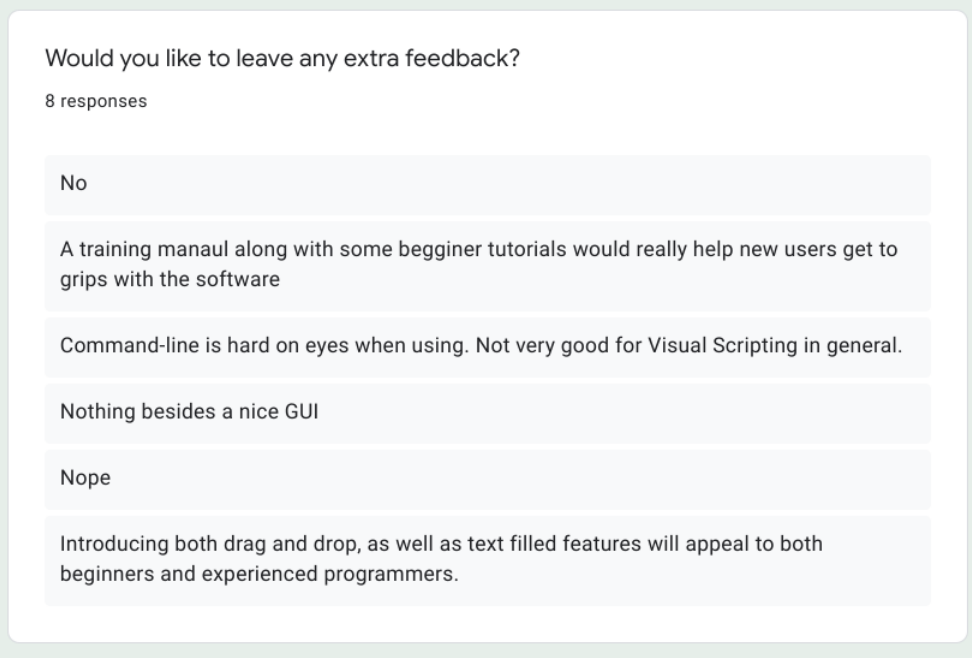

Fig. 5.27. Survey Question 19 - Found at: Original Image

6. EVALUATION OF DESIGN SURVEY FEEDBACK

By evaluating the findings of the VisualPro's initial design survey, a common understanding is that if the Visual Scripting program provides plenty of features for customisation in an 'easy to use' fashion, it will attract more attention. This conclusion comes up a few times, for instance regarding Sub-Section 5.2.1, Survey Question 1, page 24, the results states that *one out of eight responses* did not agree that either console application helps them. Previously, as mentioned, this could be an issue trying to provide the VisualPro market to a broader audience. Nonetheless, further on the survey, looking at Sub-Section 5.2.6, Survey Question 6, page 28 to Sub-Section 5.2.8, Survey Question 8, page 30, a strong indication shows that the software satisfied all participants when it comes to the UX and swiftness of creating the demanded structure that proves to execute faster than they can type it. There is a chance that the addition of GUI could create an easier-to-use environment, especially for beginners who do not enjoy the console environment. Referring directly to Sub-Section 5.2.10, Survey Question 10, page 32, all respondents agreed that a GUI would uplift the UX. This element would strongly suggest that VisualPro would likely attract beginners and existing developers if VisualPro offers a great GUI with superb User Usability with easy to access features rather than anything complicated.

Furthermore, the feedback gathers crucial information on how the eight respondents would like to see the VisualPro UI. A series of questions allows a qualitative way of seeing VisualPro's design. Data includes how the structure elements, logical elements, select languages, and saving options and sub-elements should work. After brainstorming, a node tree design approach seems to create a system that corresponds to the respondents' demands and should create an ideal visual scripting environment.

It is essential to keep quality control of VisualPro during development and get statistics from potential testers to provide significant insight on whether VisualPro is keeping things like UX, UI and User Usability on track. Any quantitative information from this survey will compare with VisualPro's final product to see if VisualPro is on track and would help VisualPro stay on track. Another thing to mention is that comparing the design survey (Mostly Qualitative Data) with the implementation survey (Mostly Quantitative Data) helps determine if VisualPro meets requirements. A method for this to work is by adding a section in both the Tutorial's and VisualPro's Implementation Survey that asks *a select* questions identically, then comparing the two Qualitative data for results.

7. DEVELOPMENT METHODOLOGY

According to ‘Choice of Software Development Methodologies: Do Organizational, Project, and Team Characteristics Matter?’ written by Leo R. Vijayasarathy, a selection of development methodologies are compared:-

- Agile Development Methodology.
- Traditional Development Methodology.
- Iterative Development Methodology.
- Hybrid Development Methodology.

Figure 7.1 by Leo R. Vijayasarathy [20] shows a comparison of projects that use what development methodology. This pie chart should help determine the best development methodology for this project.

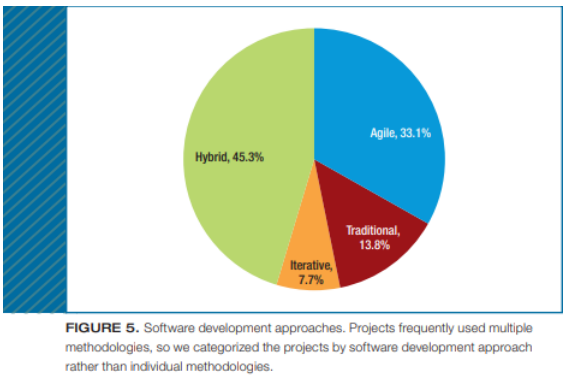

FIGURE 5. Software development approaches. Projects frequently used multiple methodologies, so we categorized the projects by software development approach rather than individual methodologies.

Fig. 7.1. Comparison of projects by development methodology. Leo R. Vijayasarathy [20]

Figure 7.2 by Leo R. Vijayasarathy [20] shows a comparison of the annual revenue and employee count of each development methodologies by the percentage of companies. This stacked bar chart should help indicate each development methodology’s an employee count by company usage; annual fees will not impact the decision to make the development methodology choice as the author states that ‘The annual revenue wasnâ€™t statistically significant; the number of employees was.’ [20]

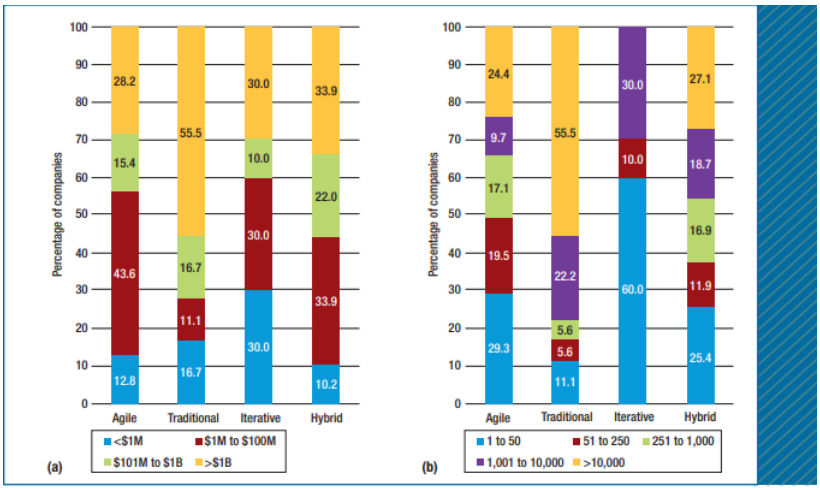

FIGURE 6. The surveyed companies according to their development approach and (a) annual revenue (US$) and (b) number of employees. The annual revenue wasn't statistically significant; the number of employees was.

Fig. 7.2. Comparison of the annual revenue and employee count of each development methodology (percentage of companies). Leo R. Vijayasarathy [20]

Figure 7.3 by Leo R. Vijayasarathy [20] shows a comparison of the budget and criticality of each development methodologies by the percentage of projects. This stacked bar chart should help indicate each development methodology's budget and criticality by project usage.

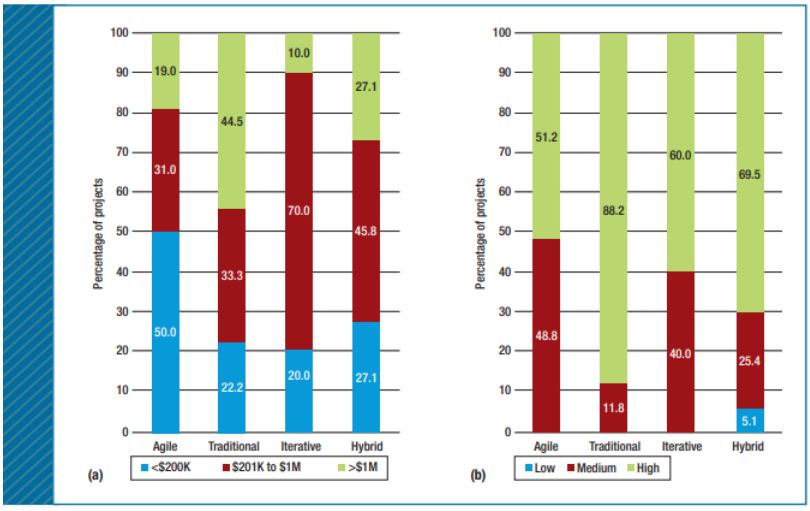

FIGURE 7. The projects according to their development approach and (a) budget and (b) criticality. Both factors had a significant association with the development approach.

Fig. 7.3. Comparison of the budget and criticality of each development methodology (percentage of projects). Leo R. Vijayasarathy [20]

After observing Figure 7.1, the Hybrid development methodology is the most used for project usage. Agile development methodology comes second, Traditional development methodology comes third, and Iterative development methodology comes fourth.

Figure 7.2 indicates that the Traditional development methodology requires the most employees to develop projects. The Iterative development methodology requires the least amount of employees to develop projects. Figure 7.3 indicates that the Agile development methodology requires less budget and is the most critical. Traditional development methodology requires the highest budget and is the least critical.

The Iterative development methodology seems to benefit this project for the following reasons:-

- Fewer employees (Fits a individual ran project).
- 60% criticality and 40% budget (Great balance).
- Well-structured development model.

This development methodology follows the following development steps:-

Mathematical Key:
$\mathcal{IP}$: Initial Planning, C: Continue?, $\mathcal{T}$: True, $\mathcal{PL}$: Planning, $\mathcal{RQ}$: Requirements
$\mathcal{AD}$: Analysis and Design, $\mathcal{IM}$: Implementation, $\mathcal{TT}$: Testing, $\mathcal{EV}$: Evaluation
$\mathcal{DP}$: Deployment

Step 1:

$$\mathcal{IP}$$

Step 2:

$$\sum_{\backslash=\infty}^{C\neq\mathcal{T}}(\mathcal{PL}(\mathcal{RQ}-> \mathcal{AD})-> \mathcal{IM}_C(\mathcal{TT}-> \mathcal{EV}))$$

Step 3:

$$\mathcal{DP}$$

The Iterative development model will start the Initial Planning, then starts the procedure by going to the Planning stage. The planning stage involves the Requirements and Analysis and Design stage. The implementation stage consists in checking if the project is ready for deployment. If this is not true, this stage involves the Testing and Evaluation stages. The deployment stage is the deployment of the product and the completion state of the project. This stage proceeds on the completion of the implementation. This process is easy to follow, and the repetition of conducting tests and reevaluating the product provides a well-tested end product. However, this process is impractical in a large team environment and can delay the product development time due to the process.

8. IMPLEMENTATION

8.1. Main Layout

Figure 8.1 is the design implementation of the VisualPro software using Visual Studio. The design starts with two panels, one containing the software's functionality and the other enabling the work area within the software. After examining the design implementation, the software lacked class, function and variable containers to create a code structure.

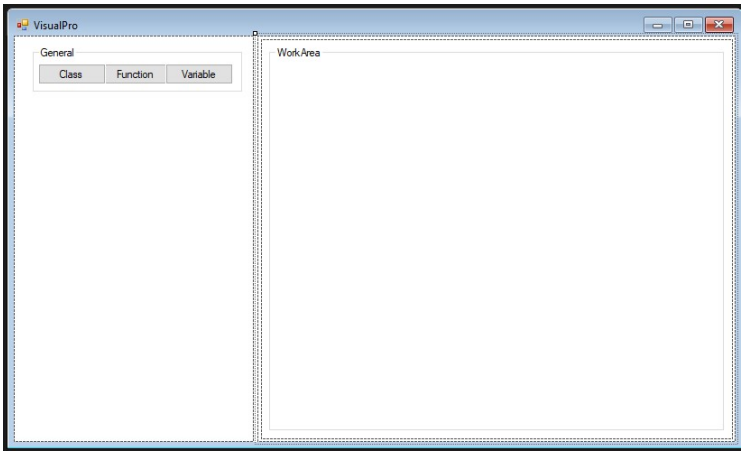

Fig. 8.1. Design Planning - Stage 1

After looking at the issues within the first stage, creating a simple class container is now available within the design implementation. However, this time the container seems too basic, and contains zero positioning ideas.

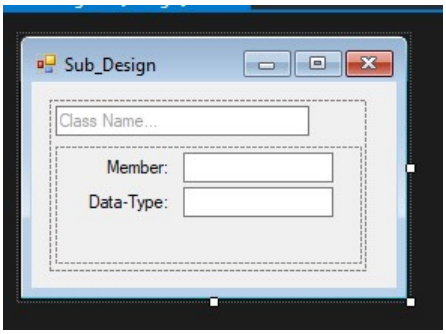

Fig. 8.2. Design Planning - Stage 2

From the previous implementation, a design of the container positions provides an idea of how the VisualPro software should enable users to drag containers on the work area. However, the design the positioning of the containers does not generate a good UI

and UX design. Another design problem, the software has no options to save the code when the user creates a new project. An extra point to make is that the containers should have access for children and any additional properties.

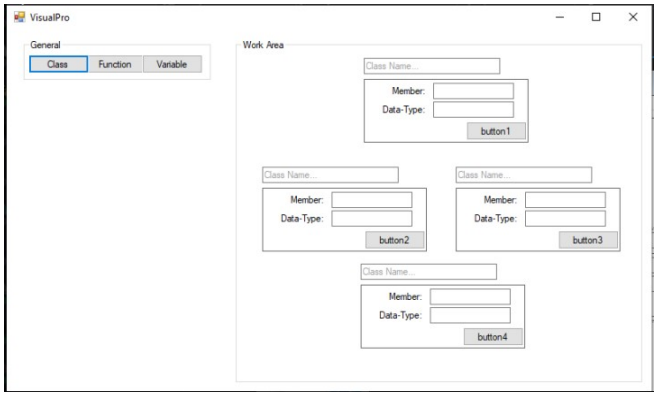

Fig. 8.3. Design Planning - Stage 3

The previous problem of the last implementation design contains no option to save is now available. The parent and child containers need remaking with a new position scheme for the next iteration. A foreseeable design flaw of Classes, Functions and Variables is that children cannot be higher than their inheritance type during development under the parent container. This problem meant, for example, that the class would be declared inside another class programmatically if the problem occurred under the current logic when the compilation happened. The code syntax makes no sense, resulting in a compilation error. An example of this error within this software could be an *'int variable = int variable;'*.

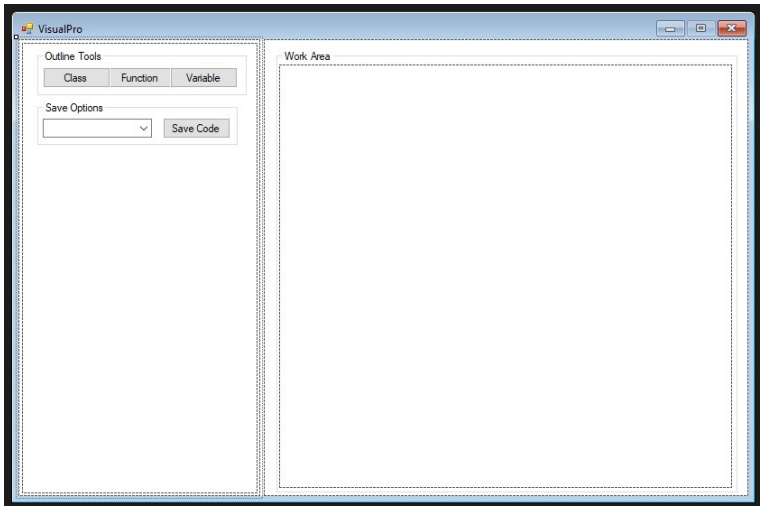

Fig. 8.4. Design Planning - Stage 4

Figure 8.5 shows the implementation of the design using the designs shown beforehand. Rather than shortcutting relationships between each class, function and variable by nesting them, the final implementation design prevents the user from dragging and dropping the same type of container onto another container. A property button appears on each

container to control function arguments and relationships between classes, functions, and variables, opening up a dialogue containing a property sheet. If any error occurs, there are no error messages or dialogue box designs to alert the user if something is not working as it should be.

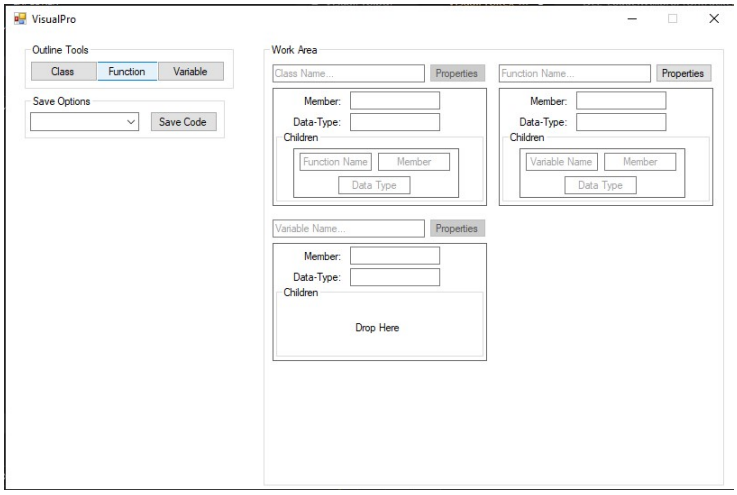

Fig. 8.5. Design Planning - Stage 5

8.2. Error Handling

The previous iteration implementation design problem, ‘Error Dialogue,’ is now available. This dialogue design helps inform the user if the software fails. The dialogue only activates if there is a syntax language error within the ‘VisualPro.xml’ language configuration file. This design should expand to include other errors that occur within the software.

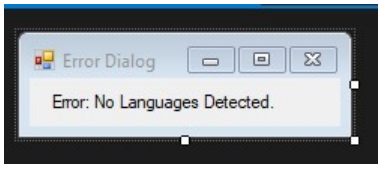

Fig. 8.6. Error Handling

9. EVALUATION OF FUNCTIONALITY

During the development of VisualPro, a few unit tests to test mainly the backend would give some idea of evaluation of the functionality. These tests will cross-reference with the front-end implementation to establish whether the lack of functionality comes down to the front-end or the backend implementation. Any bugs found should list within the tutorial to help any new developers navigate VisualPro without hitting the same known bugs.

9.1. Decoder Tools

Tools to parse encrypted strings of characters full of information to decipher particular information help the communication between the front-end and the backend. In some cases, tool functionality developed within C# for the VisualPro software seems to slow down the software, so in extreme cases, redevelopment with the C++ language of these tools to use in the VisualPro software increased the performance of the software.

'FindSubStrA' is a test that searches for a word between two characters. The function (originally in C#) asks for a start **(char)**, end **(char)**, position **(int)**, and a string **(std::string)** to make this search possible. All the tests within 'FindSubStrA' are successful.

```
1         TEST_METHOD(FindSubStrA)
2         {
3           /* Error Log:
4           * All tests passed.
5           * Purpose of tool is to find a word between two characters.
6           */
7           // Seperators: | |
8           Assert::AreEqual("H", CSFindSubStr('|', '|', 0,
                "|H|E-L-LxO").c_str());
9
10          // Seperators: | -
11          Assert::AreEqual("E", CSFindSubStr('|', '|', 1,
                "|H|E|L-LxO").c_str());
12
13          // Seperators: - -
14          Assert::AreEqual("L", CSFindSubStr('-', '-', 0,
                "|H|E-L-LxO").c_str());
15
16          // Seperators: - -
17          Assert::AreEqual("L", CSFindSubStr('-', 'x', 1,
                "|H|E-L-LxO").c_str());
18
```

```
19            // Seperators: - \0
20            Assert::AreEqual("O", CSFindSubStr('x', '\0', 0,
                  "|H|E-L-LxO").c_str());
21          }
```

'FindSubStrB' tests the C++ version of the previous function. This time it checks for a separator and position. The arguments to find a separated substring, string **(const char*)**, string to test **(const char*)**, separator **(char)** and position **(int)**. One test passed, and the other two failed. This test problem is one of the reasons why the code generation does not generate properly. Without this test, it would have been hard to decipher the survey code submitted further down.

```
1           TEST_METHOD(FindSubStrB)
2           {
3             /* Error Log:
4             * Test A - Failed -> Test Written Wrongly | Passed
5             * Test B - Passed
6             * Test C - Failed -> Selects Each Word | Temporary Solution: Read
                  Below.
7             */
8
9             // Seperator |
10            // Checks all words. - Test A
11            for (size_t i = 0; i < 3; i++)
12              Assert::AreEqual(("Word " + std::to_string(i + 1)).c_str(),
                    CPPFindSubStr("Word 1|Word 2|Word 3", "Word", '|', i).c_str());
13
14            // Finds First Word - Test B
15            // Seperator x
16            Assert::AreEqual("A", CPPFindSubStr("AxYZxQRxAxB", "A", 'x',
                  0).c_str());
17
18            // Finds First Word in Each Position - Test C
19            // Seperator x
20
21            // current status: Fail (selects each word) - Temporary solution:
22            // if(word == "A") or further rework
23            std::string word = std::string();
24            for (size_t i = 0; !(word = CPPFindSubStr("AxYZxQRxAxB", "A", 'x',
                  (int)i)).empty(); i++)
25              if (word == "A")
26                Assert::AreEqual("A", word.c_str());
27          }
```

'FindStrIndexA' test will observe the substring index from a set of arguments. The functionality enables the developer to find an index of a string's character. The function requires two arguments, string **(const char*)** and test subject **(char)**. All parts of the test passed.

```
        TEST_METHOD(FindStrIndexA)
        {
          /* Error Log:
          * Test A - Passed
          * Test B - Passed
          * Test C - Passed
          */

          //Expectation: 6 | Test A
          Assert::AreEqual(6, FindStrIndex("!)2pAL@sSwS", '@'));
          //Expectation: Not -1 && is 8 | Test B
          Assert::AreNotEqual(-1, FindStrIndex("!)2pAL!sSwS", 'S'));
          Assert::AreEqual(8, FindStrIndex("!)2pAL!sSwS", 'S'));


          // Expectation: -1 | Test C
          Assert::AreEqual(-1, FindStrIndex("!)2pAL!sSwS", '~'));
          Assert::AreEqual(-1, FindStrIndex("!)2pAL!sSwS", '1'));
          Assert::AreEqual(-1, FindStrIndex("!)2pAL!sSwS", 'M'));
        }
```

'ReplaceWordA' tests the replace string by a substring. The function requires five arguments, string **(const char*)**, depth **(int)**, start **(char)**, end **(char)** and replaceString **(const char*)**. All tests passed.

```
        TEST_METHOD(ReplaceWordA)
        {
          /* Error Log
          *
          */
          const std::string aMistake =
              "A:Banana\x1f\B:Pineapple\x1f\C:Grapefruit",
            testA = ReplaceWord(aMistake.c_str(), 1, ':', '\x1f', "Apple"),
            testB = ReplaceWord(testA.c_str(), 3, ':', '\x1f', "Banana");
          const char* expected = "A:Apple\x1f\B:Pineapple\x1f\C:Grapefruit",
               *secExpected = "A:Apple\x1f\B:Pineapple\x1f\C:Banana";

          Assert::AreNotEqual(expected, aMistake.c_str());
          Assert::AreEqual(expected, testA.c_str());
          Assert::AreEqual(secExpected, testB.c_str());
        }
```

9.2. Lists

VisualPro communicates to the Programming Planner library using C++ 'Lists' class. Without the 'List' class functionality testing, many errors that could be down to the 'Lists'

class or front-end implementation would make this project impossible to create. Three important functionality of 'Lists' class are the 'GetSignature', 'Addition' and 'Deletion' functions. The signature signifies whether an item is a Class, Function, Argument or Variable to the Programming Planner.

The 'List_Addition' enables the addition of Lists. The test examines the addition of an item and passes the first try. When creating the VisualPro software, any communication issues would lay on the front-end side.

```
1         TEST_METHOD(List_Addition)
2         {
3           /* Error Log:
4             First Try: Pass - (A, B, C, D, E, F, G, H)
5           */
6
7           // Creation of a new list with X, on addition the ID's should follow
                as X-1, X-2 etc.
8           List* xList = new List("X");
9           List* yList = new List("Y"); // Same but Y-1, Y-2 etc.
10          List* xyList = new List("XY"); // XY-1, XY-2 etc.
11
12          // Should return X - Test | A
13          Assert::AreEqual("X", xList->GetSignature().c_str());
14
15          // Should return Y - Test | B
16          Assert::AreEqual("Y", yList->GetSignature().c_str());
17
18          // Should return XY - Test | C
19          Assert::AreEqual("XY", xyList->GetSignature().c_str());
20
21          // Should return X-1 - Test | D
22          Assert::AreEqual("X-1", xList->Add("parentItem"));
23
24          // Should return Y-1\x1fP-X-1 - Test | E
25          Assert::AreEqual("Y-1\x1fP-X-1", yList->Add("childItem", "X-1"));
26
27          // Should return Y-2 - Test | F
28          Assert::AreEqual("Y-2", yList->Add("parentItem"));
29
30          // Should return X-2 - Test | G
31          Assert::AreEqual("X-2", xList->Add("parentItem"));
32
33          // Should return XY-1 - Test | H
34          Assert::AreEqual("XY-1\x1fP-X-2", xyList->Add("parentItem", "X-2"));
35
36          // Garbage collection for the test.
37          delete xList; delete yList; delete xyList;
38        }
```

VisualPro tries to offer as much functionality as possible, including container removal. This functionality will not work if the code generator generates the same items, where 'List_Deletion' comes into play. The 'List_Deletion' tests the list deletion. The test passes the first try.

```
        TEST_METHOD(List_Deletion)
        {
          /* Error Log:
          * First Try: Pass
          */

          // Creation of X
          List* list = new List("X");
          list->Add("Function 1"); // ID: X-1
          list->Add("Function 2"); // ID: X-2
          list->Add("Function 3"); // ID: X-3

          // Both work but top is more tidy and saves some memory
          // History: Design of List was originally for displaying on Console and therefore saves any extra logic after calling ListAll method.
          // Ideal - Test | A-I:
          Assert::AreNotEqual("X-1 Function 1\nX-2 Function 2\nX-3 Function 3", list->ListAll().c_str());
          // Actual - Test | A-A:
          Assert::AreEqual("X-1 Function 1 \nX-2 Function 2 \nX-3 Function 3 \n\n", list->ListAll().c_str());

          list->Remove(2); // Expectation: Function 2 goes disappears

          // To test if removal of List works - Test | B
          Assert::AreEqual("X-1 Function 1 \nX-3 Function 3 \n\n", list->ListAll().c_str());

          // Performance Improvement
          // Ideal - It would be ideal to replace a empty ID (X-2 is non-existent, though X-4 is next) - Test | C-I
          Assert::AreNotEqual("X-2", list->Add("Method 2")); // The list should fill in the empty ID to save data.

          Assert::AreEqual("X-1 Function 1 \nX-3 Function 3 \nX-4 Method 2 \n\n", list->ListAll().c_str());

          // However, if the last element is non existence, the ID will rollback to any previously unique ID's that are not taken by the last element.)
          // Which, in theory provides short IDs that are suitable for Programming Planner's solution.
          // Actual - Test | D-A
          list->Remove(2);
          list->Remove(3);
```

```
36            list->Remove(4);
37            Assert::AreEqual("X-2", list->Add("Method 2"));
38
39            delete list;
40          }
```

10. EVALUATION OF FEEDBACK

10.1. Programmer Planner vs. VisualPro

Figure 10.1: Two respondents, 33.3%, have experience with Programming Planner previously. These candidates will help compare the two programs. Within the first question, 'What program delivered better user experience?' - zero being Programming Planner and ten being VisualPro. The two respondents range from option nine to ten, indicating that they prefer VisualPro to Programming Planner as a development tool.

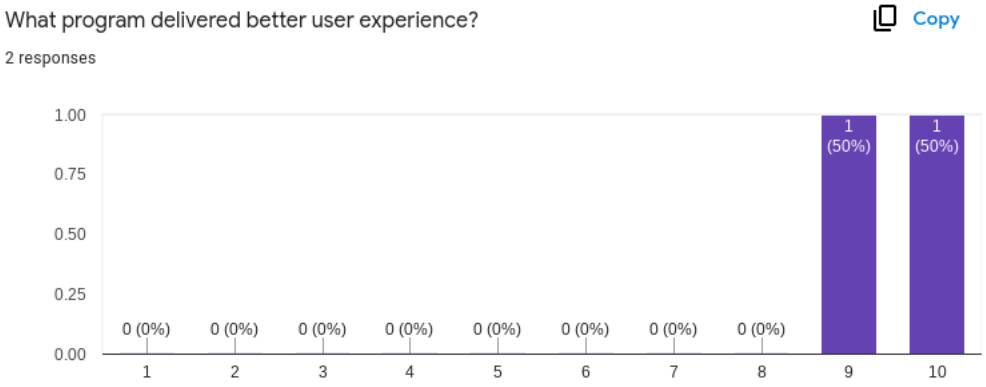

Fig. 10.1. Evaluation of Feedback - SQ-A1

Figure 10.2: 'Which software ticks your boxes for a development tool?', has four attributes, 'Ease of Use', 'Performance', 'Extra Features' and 'File Structure', and allows the respondents to choose Programming Planner or VisualPro. The key within figure 10.2 indicates whether the respondents chose Programming Planner or VisualPro. The respondents chose VisualPro for each attribute.

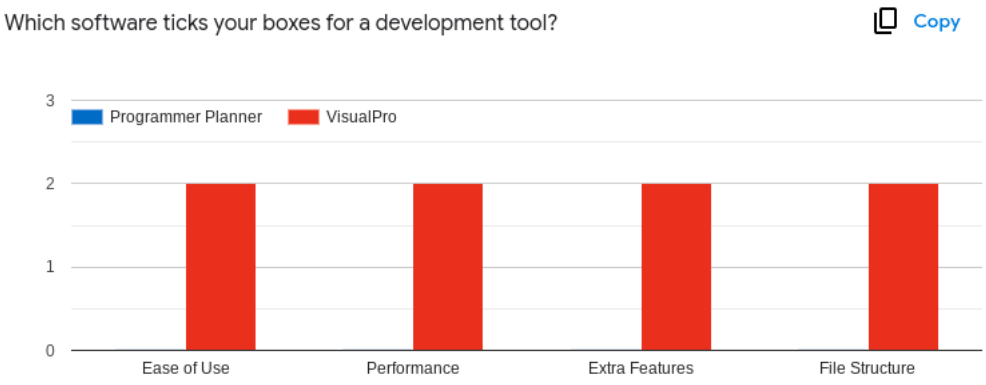

Fig. 10.2. Evaluation of Feedback - SQ-A2

Figure 10.3: ‘Does VisualPro improve the user experience instead of Programming Planner?’, all respondents chose ‘Yes’, displaying that the respondents feel VisualPro made improvements to the UX from the Programming Planner software.

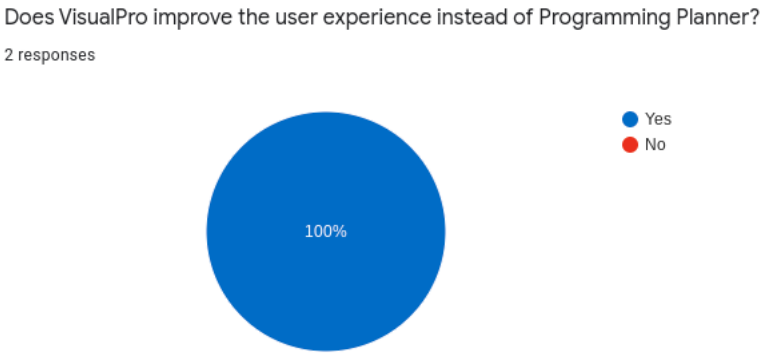

Fig. 10.3. Evaluation of Feedback - SQ-A3

Figure 10.4: ‘Which program is more beginner friendly?’ measured on a scale of one (Programming Planner) to ten (VisualPro). Half of the respondents chose eight, and the other half chose ten. This result suggests that VisualPro is more beginner-friendly than Programming Planner.

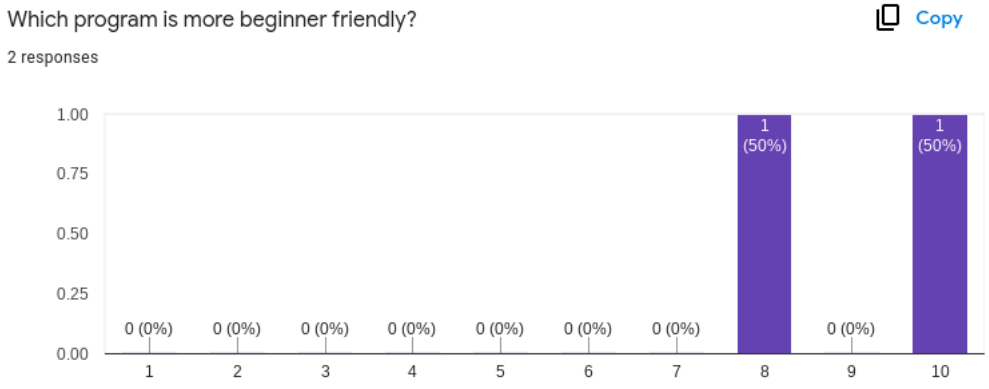

Fig. 10.4. Evaluation of Feedback - SQ-A4

Figure 10.5: ‘Does the software provide enough tuitive ways to create structure?’ half of the respondents chose ‘Yes’, and the other half chose ‘No’. This response demonstrates that the software could do with more functionality to provide a better-structured environment with remarkable UX and User Usability.

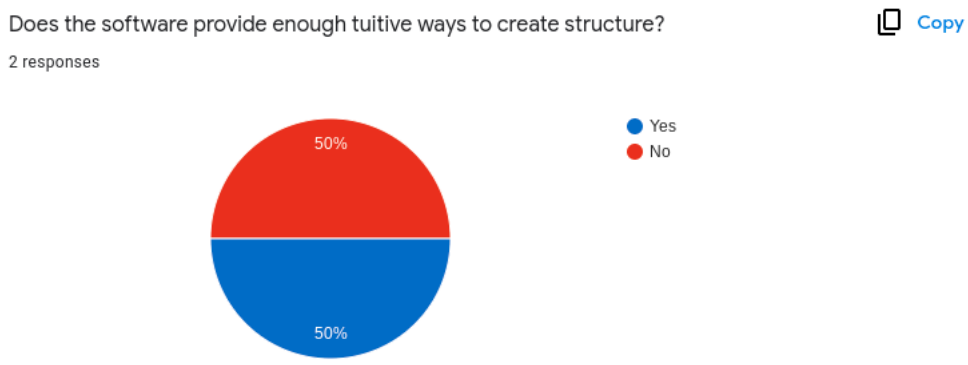

Fig. 10.5. Evaluation of Feedback - SQ-A5

'List any unknown bugs and provide steps', listed responses:

- 'The end code that was generated need more configuaration.'
- 'havent found any'.

10.2. VisualPro

This section includes all the six respondents. Two of these respondents have prior experience with testing VisualPro and Programming Planner. The marking scheme analyses if the code generation is correct to the example shown within the tutorial, partially correct; due to bugs or other issues or not correct. This data would be passed through a Fuzzy Expert Logic algorithm to determine the overall User Usability.

10.2.1. Object-Oriented Programming - Tutorial A | Exercise: Creating a Class

This section of Tutorial A aims to create a' World' class. The initial question of this section, 'Write your understanding of a Class from what you have learnt from Tutorial A', listed responses:

- 'I have learnt and used the class as a container and used it with ease in VisualPro.'
- 'A way to contain code to use later on'.
- 'a class enables developers to containerise code that can be referenced later as an object.'
- 'A class is code that is containerised so it can be referenced at a later stage as an object.'
- 'Classes provide developers with the ability to use a string syntax in object-oriented environments'.

- ‘A way to group blocks of code’.

Figure 10.6: ‘Rate the ease of creating a Class within VisualPro’ on a one (understandable) to ten (Incomprehensible) scale. 33.3% of the respondents chose one, 16.7% chose two, 33.3% chose three, and 16.7% chose five. This data could imply that the structure is not quite there yet, and it is essential to focus on the UX and User Usability of the existing features before adding more.

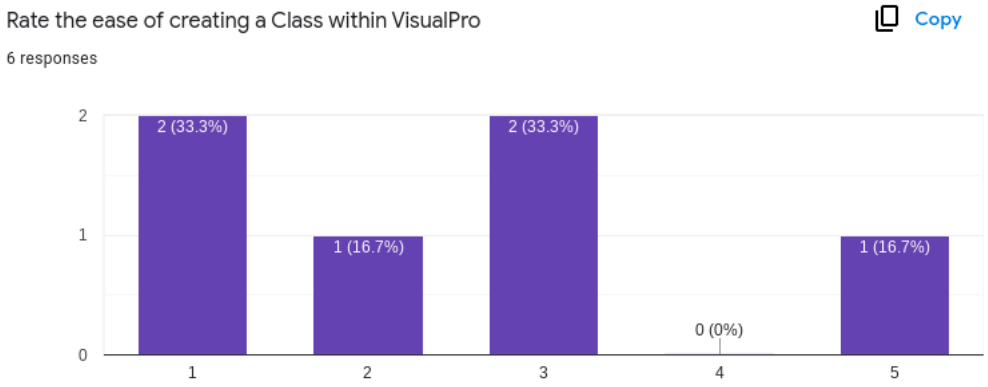

Fig. 10.6. Evaluation of Feedback - SQ-BAA1

Figure 10.7: ‘Did your VisualPro Work Area look identical to the example shown?’, 66.7% respondents chose ‘Yes’, 33.3% chose ‘No’. This response points out that four code generations are correct and the others are not.

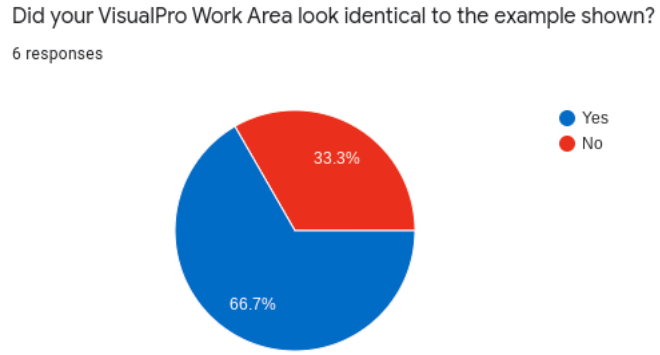

Fig. 10.7. Evaluation of Feedback - SQ-BAA2

‘Your Code Snippet’, listed responses:

First Response:

```
        #include <iostream>
        #include <string>

        class name
```

```
{
  class-1
};
```

- This respondent seems to have followed this task correctly.

Second Response:

```
#include <iostream>

#include <string>

class name
{
  class-1
};
```

- This respondent seems to have followed this task correctly.

Third Response:

```
class World {

};
```

- This respondent followed this task correctly.

Fourth Response:

```
class Name {
  protected :
    void Method () {}
  public :
    void Method () {}
  private :
    void Method () {}
};
```

- This respondent followed this task incorrectly.

Fifth Response:

```
#include <iostream>
#include <string>

class World
{
  class-1
};
```

- This respondent followed this task correctly.

Sixth Response:

```
#include <iostream>
#include <string>

class name
{
  class-1
};
```

- This respondent seems to have followed this task correctly.

10.2.2. Object-Oriented Programming - Tutorial A | Exercise: Animal Types

The aim of this tutorial is to create different Animal classes to get use to OOP style structures. The question 'What main methods/variables separate Reptiles, Mammals and Amphibians from one another?' aims to test the respondents ability to select the correct details within OOP style prorgamming. The following responses were given:

- 'amimal type?'.
- 'sound'.
- 'Class'.
- 'Method Name'.

Two out of four responses chose a reasonable answer; these are 'animal type?' and 'sound'. The last two may or may not have confused the question and tried to answer the question by OOP terminology. The responses are fair and may suggest that the tutorial helped the respondents to understand OOP style programming or think of a different way of describing OOP style programming. Two out of four responses chose a reasonable answer; these are 'animal type?' and 'sound'. The last two may or may not have confused the question and tried to answer the question by OOP terminology.

Figure 10.8: 'Did your VisualPro Work Area look identical to the example shown?', 66.7% respondents chose 'Yes', 33.3% chose 'No'. This response points out that four code generations are correct and the others are not.

'Your Code Snippet', listed responses:

First Response:

```
class Reptile {

};
```

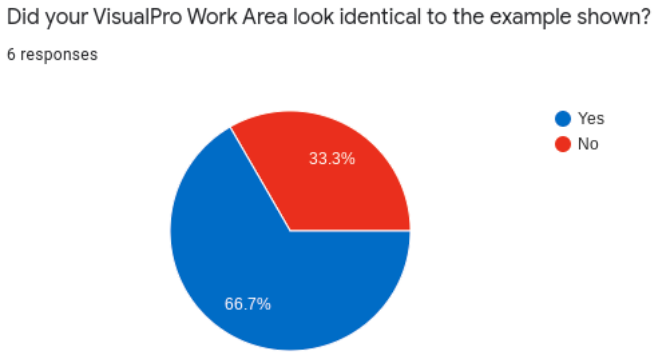

Fig. 10.8. Evaluation of Feedback - SQ-BAB1

```
4
5         class Mammals {
6
7         };
8
9         class Amphibian {
10
11        };
```

- This respondent followed this task correctly.

Second Response:

```
1         #include <iostream>
2         #include <string>
3
4         class name
5         {
6           class-1
7         };
8
9         class name
10        {
11          class-2
12        };
13
14        class name
15        {
16          class-3
17        };
```

- This respondent seems to have followed this task correctly.

Third Response:

```
1         #include <iostream>
2         #include <string>
```

```
class name
{
  class-1
};

class name
{
  class-2
};

class name
{
  class-3
};

class name
{
  class-4
};
```

- This respondent seems to have followed this task correctly.

Fourth Response:

```
#include <iostream>
#include <string>

class name
{
  class-1
};

class name
{
  class-2
};

class name
{
  class-3
};
```

- This respondent seems to have followed this task correctly.

Fifth Response:

```
class Name {
protected :
  void Method () {}
```

```
        public :
          void Method () {}
        private :
          void Method () {}
        };
```

- This respondent followed this task incorrectly.

Sixth Response:

```
        #include <iostream>
        #include <string>

        class name
        {
          class-1
        };

        class name
        {
          class-2
        };

        class name
        {
          class-3
        };
```

- This respondent seems to have followed this task correctly.

10.2.3. Tutorial A | Exercise: Vehicle Components

The objective of this exercise is to ask the respondents to think of a few vehicle components and their corresponding methods, whilst implmentating the product into VisualPro. Figure 10.9: 'Did your VisualPro Work Area look identical to the example shown?', 66.7% respondents chose 'Yes', 33.3% chose 'No'. This response points out that four code generations are correct and the others are not.

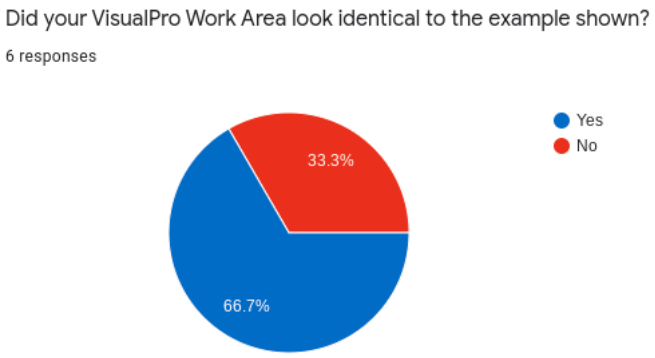

Fig. 10.9. Evaluation of Feedback - SQ-BAC1

‘Your Code Snippet’, listed responses:

First Response:

```
        class Engine {
        };

        class Wheel {
        };
```

- This respondent followed this task correctly.

Second Response:

```
        #include <iostream>

        #include <string>

        dt name()
        {
          function-1
        }

        dt name
```

- This respondent followed this task incorrectly.

Third Response:

```
        #include <iostream>
        #include <string>

        class name
        {
          class-1
        };

```

```
          class name
          {
            class-2
          };

          class name
          {
            class-3
          };

          class name
          {
            class-4
          };

          class name
          {
            class-5
          };

          dt namedt namedt namedt name
```

- This respondent seems to have followed this task correctly.

Fourth Response:

```
          Public class name
          {
            class-1
          }

          dt name()
          {
            function-1
          }

          dt name()
          {
            function-2
          }

          dt name()
          {
            function-3
          }
```

- This respondent seems to have followed this task correctly.

Fifth Response:

```
.
```

- This respondent followed this task incorrectly.

Sixth Response:

```
member class name
{
  class-1
}

member class name
{
  class-2
}

member class name
{
  class-3
}

dt name()
{
  function-1
}

dt name()
{
  function-2
}

dt name()
{
  function-3
}

dt name()
{
  function-1
}

dt namedt name
```

- This respondent seems to have followed this task correctly.

10.2.4. Object-Oriented Programming - Tutorial A | Exercise: Try It Yourself (TIY)

This exercise allows the respondents to get creative and try creating some OOP scenarios of their own. Figure 10.10: ‘Did your VisualPro Work Area look identical to the example

shown?', 66.7% respondents chose 'Yes', 33.3% chose 'No'. This response points out that four code generations are correct and the others are not.

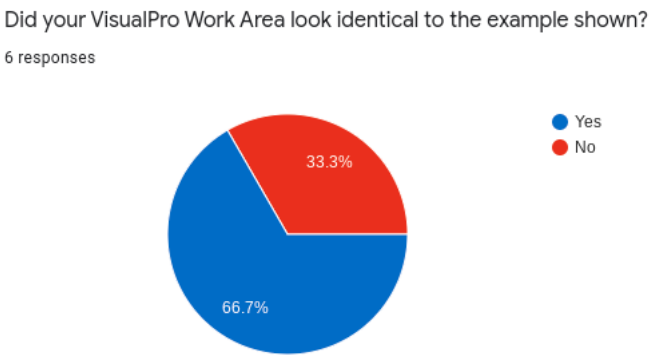

Fig. 10.10. Evaluation of Feedback - SQ-BAD1

'Your Code Snippet', listed responses: *First Response:*

```
        class Planet {
        };

        class Computer {
        };
```

- This respondent followed this task correctly.

Second Response:

```
        #include <iostream>
        #include <string>

        class Computer
        {
          class-1
        };
```

- This respondent followed this task correctly.

Third Response:

```
        n/a
```

- This respondent followed this task incorrectly.

Fourth Response:

```
        using System;

        public class name
```

```
        {
        class-1
        }

        dt name
```

- This respondent followed this task correctly.

Fifth Response:

```
        .
```

- This respondent followed this task incorrectly.

Sixth Response:

```
        member class name
        {
          class-1
        }

        member class name
        {
          class-2
        }

        member class name
        {
          class-3
        }

        dt name()
        {
          function-1
        }

        dt name()
        {
          function-2
        }

        dt name()
        {
          function-3
        }

        dt name()
        {
          function-1
        }
```

```
35
36          dt namedt name
```

- This respondent seems to have followed this task correctly.

10.2.5. Functional Programming - Tutorial B | Exercise: Toaster Functionality

'Toaster Functionality' exercise is to establish the functionality of a toaster to get grips of FP style programming. The initial question, 'What is the difference between OOP and FP styles?' helps understand if the respondents' have learnt the complexity of the two styles. Responses of the respondents:

- 'I personally found less code but the oop style is easier to read.'
- 'oop focuses on user objects over functional approach'.
- 'not sure'.
- 'FP uses a fixed data structure, while OOP applies a variable one.'
- 'OOP enables object creation. FP style contains functional structure.'
- 'OOP is more dynamic compared to the FP which is more structured'.

- Generally speaking, the respondents have learned the differences between both styles throughout the tutorials and VisualPro experience.

Figure 10.11: 'Was creating an FP style structure than traditional methods?', 83.3% respondents chose 'Yes', 16.7% chose 'No'. This question tries to figure out if using VisualPro makes it easier implementating FP style programming over traditional methods. The responses indicate that VisualPro seems to make structure programming easier over typing out structure in FP style. However, bare in mind that these respondents may or may have not touched any type of programming language previously.

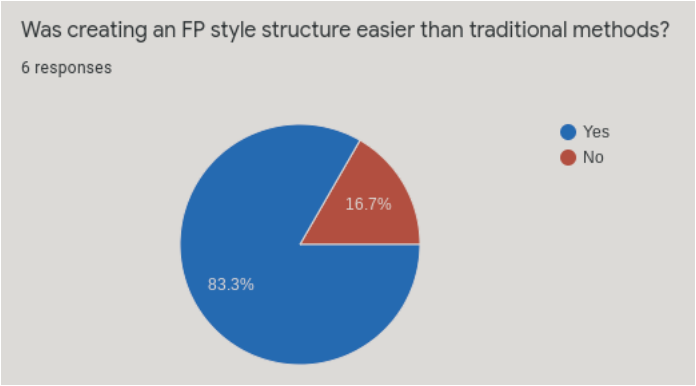

Fig. 10.11. Evaluation of Feedback - SQ-BBA1

Figure 10.12: ‘Did your VisualPro Work Area look identical to the example shown?’, 66.7% respondents chose ‘Yes’, 33.3% chose ‘No’. This response points out that four code generations are correct and the others are not.

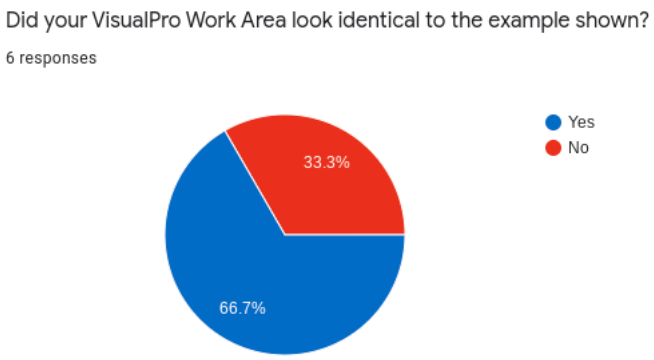

Fig. 10.12. Evaluation of Feedback - SQ-BBA2

‘Your Code Snippet’, listed responses:

First Response:

```
        #include <iostream>

        #include <string>

        dt name()
        {
          function-1
        }

        dt name()
        {
          function-2
        }

        dt name()
        {
          function-3
        }
```

- This respondent seems to have followed this task correctly.

Second Response:

```
        not sure
```

- This respondent followed this task incorrectly.

Third Response:

```
          class name
          {
            class-1
          };

          class name
          {
            class-2
          };

          class name
          {
            class-3
          };

          class name
          {
            class-4
          };

          class name
          {
            class-5
          };

          dt name()
          {
            function-1
          }

          dt name()
          {
            function-2
          }

          dt name()
          {
            function-3
          }

          dt namedt namedt namedt name
```

- This respondent followed this task incorrectly.

Fourth Response:

```
          # include < stdio .h >
          // Declaration
          float XMultiplyY ( double x , double y ) ;
          int main (int argc , char ** argv ) {
```

```
        printf (" 8.87 * 2.65 = %f", XMultiplyY (8.87 , 2.65) ) ;
        // result : 23.505501
        return 0;
        }
        // Definition
        float XMultiplyY ( double x , double y ) {
        return x * y
```

- This respondent followed this task incorrectly.

Fifth Response:

```
        #include <iostream>
        #include <string>

        dt name()
        {
          function-1
        }

        dt name()
        {
          function-2
        }

        dt name()
        {
          function-3
        }
```

- This respondent seems to have followed this task correctly.

Sixth Response:

```
        #include <iostream>
        #include <string>

        dt name()
        {
          function-1
        }

        dt name()
        {
          function-2
        }

        dt name()
        {
          function-3
```

```
17          }
```

- This respondent seems to have followed this task correctly.

10.2.6. Functional Programming - Tutorial B | Exercise: Toaster Variables

The purpose of this exercise is to test the respondents on the ability of selecting Toaster Variables. These can vary from temperature, timers and other variables. Figure 10.13: 'Did your VisualPro Work Area look identical to the example shown?', 66.7% respondents chose 'Yes', 33.3% chose 'No'. This response points out that four code generations are correct and the others are not.

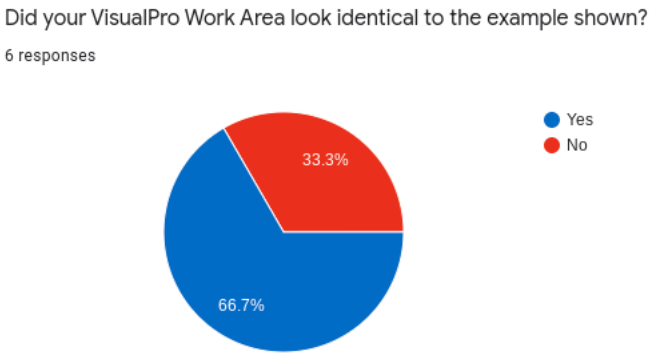

Fig. 10.13. Evaluation of Feedback - SQ-BBB1

'Your Code Snippet', listed responses:

First Response:

```
1           #include <iostream>
2           #include <string>
3
4           dt namedt name
```

- This respondent seems to have followed this task correctly.

Second Response:

```
1           #include <iostream>
2           #include <string>
3
4           dt name()
5           {
6             function-1
7           }
8
9           dt name()
```

```
          {
            function-2
          }

          dt name()
          {
            function-3
          }
```

- This respondent followed this task incorrectly.

Third Response:

```
          #include <iostream>
          #include <string>

          dt name()
          {
            function-1
          }

          dt namedt namedt name
```

- This respondent seems to have followed this task correctly.

Fourth Response:

```
          class name
          {
            class-1
          };

          class name
          {
            class-2
          };

          class name
          {
            class-3
          };

          class name
          {
            class-4
          };

          class name
          {
            class-5
```

```
        };

        dt name()
        {
          function-1
        }

        dt name()
        {
          function-2
        }

        dt name()
        {
          function-3
        }

        dt namedt namedt namedt name
```

- This respondent followed this task incorrectly.

Fifth Response:

```
        float temperature ;
        int timer;
```

- This respondent followed this task correctly.

Sixth Response:

```
        #include <iostream>
        #include <string>

        dt namedt name
```

- This respondent seems to have followed this task correctly.

10.2.7. Functional Programming - Tutorial B | Exercise: Fuel Station

The 'Fuel Station' exercise provides some direction. As long as the respondent can identify ***any*** functions or variables within a Fuel Station indicates that they are following the task correctly. Figure 10.14: 'Did your VisualPro Work Area look identical to the example shown?', 66.7% respondents chose 'Yes', 33.3% chose 'No'. This response points out that four code generations are correct and the others are not.

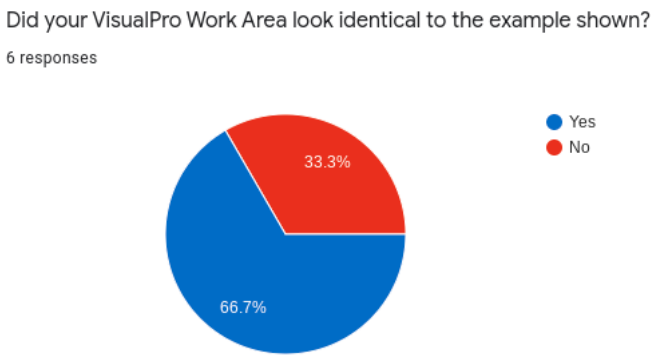

Fig. 10.14. Evaluation of Feedback - SQ-BBC1

‘Your Code Snippet’, listed responses:

First Response:

```
        #include <iostream>
        #include <string>

        dt name()
        {
          function-1
        }

        dt name()
        {
          function-2
        }

        dt name()
        {
          function-3
        }

        dt name
```

- This respondent seems to have followed this task correctly.

Second Response:

```
        double Cashier () {
          int availablePumps ;
          return 0.00;
        }
        int GetAvailablePumps () { }
        double GetFuelLevel () { }
```

- This respondent followed this task correctly.

Third Response:

```
#include <iostream>
#include <string>

dt name()
{
  function-1
}

dt name()
{
  function-2
}

dt name()
{
  function-3
}
```

- This respondent seems to have followed this task correctly.

Fourth Response:

```
#include <iostream>
#include <string>

dt name()
{
  function-1
}

dt namedt namedt name
```

- This respondent seems to have followed this task correctly.

Fifth Response:

```
#include <iostream>
#include <string>
```

- This respondent seems to have followed this task incorrectly.

10.2.8. Functional Programming - Tutorial B | Exercise: Try It Yourself (TIY)

The TIY exercise enables the respondents to show off their new skillsets to see how effective the tutorial and VisualPro process was. Figure 10.15: 'Did your VisualPro Work Area look identical to the example shown?', 66.7% respondents chose 'Yes', 33.3% chose

'No'. This response points out that four code generations are correct and the others are not.

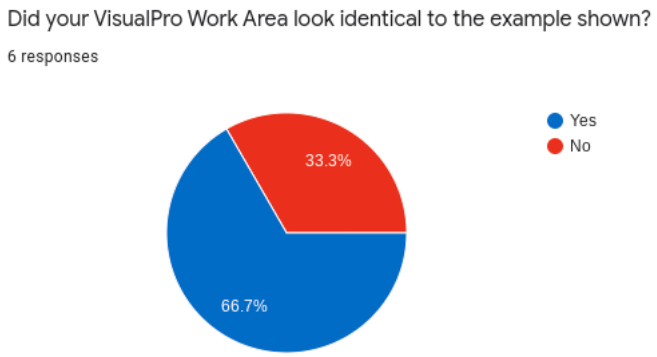

Fig. 10.15. Evaluation of Feedback - SQ-BBD1

'Your Code Snippet', listed responses:

First Response:

```
        #include <iostream>
        #include <string>
```

- This respondent seems to have followed this task incorrectly.

Second Response:

```
        #include <iostream>
        #include <string>

        dt name()
        {
          function-1
        }

        dt namedt namedt name
```

- This respondent seems to have followed this task correctly.

Third Response:

```
        n/a
```

- This respondent followed this task incorrectly.

Fourth Response:

```
        #include <iostream>
        #include <string>

```

```
          dt name()
          {
            function-1
          }

          dt name()
          {
            function-2
          }

          dt name()
          {
            function-3
          }

          dt namedt namedt name
```

- This respondent seems to have followed this task correctly.

Fifth Response:

```
          .
```

- This respondent followed this task incorrectly.

Sixth Response:

```
          #include <iostream>
          #include <string>
```

- This respondent seems to have followed this task incorrectly.

10.3. Video Footage

Two respondents' submitted video feedback. One contains audio, whereas the other clip does not. To analyse the feedback, recommendations by Dana Chisnell [3], Usefulness, Efficiency, Effectiveness, Learnability and Satisfaction categories help identify critical areas within the User Usability of VisualPro.

Displays of hesitation, missteps, experimentation and ease of use are looked at to determine any flaws within the User Usability. When any audio is present in the footage, vocal interactions like feedback, frustration, confidence and assertiveness resolve any deficiencies within the User Usability.

Respondent A demonstrates excellent understanding, confidence and fluency when using the VisualPro program. Although the respondent did show frustration regarding the code generation section, the respondent did not quit the software, trying different

attempts to resolve the problem. This scenario happened is to a known issue mentioned in the Tutorial. Respondent B finds it hard to understand the file extension and how the generated file should open to read the code content. The respondent used VisualPro by adding extra functionality to their code project. Some feedback from Respondent B is that things like deleting child containers and finding it hard to identify what container types were in the 'Work Area'. The fact the respondent pinned the VisualPro software to their taskbar implies further use of the software. Respondent B noted 'I may come back to this at some point as I did find it interesting', which shows that the software has potential in a development environment. Respondent B also experiments with keyboard shortcuts to navigate the software, whereas Respondent A only used mouse control.

Monitoring the respondents' quickness, accuracy, and confidence will justify the user's efficiency in using the program. This analysis identifies any problematic features located within the software for further development. Respondent A seemed to perform tasks quickly and accurately throughout the video footage. Respondent B also uses the program swiftly and with accuracy. This respondent also comments 'snappy' describes the efficiency. This correlation shows that the software provides an intuitive UI that efficiently enables users to create a code structure.

To work out the effectiveness is to ensure that the respondents can complete tasks with remarkable results each time. Any known bugs will not distinguish the final result. After examining both respondents, they both executed the lessons well and tried to experiment with the software, keeping on the lines of the tutorial tasks. The data indicates that the software effectively produces and generates the desired code.

To establish the learnability of the VisualPro software, the following points are looked into:-

- **Proficiency**: How well the respondents' perform as the tutorial progresses?
- **Cognitive Ability**: Observation of the respondents' ability to problem solve in a restricted environment.
- **Creativity and Innovation**: The respondents' ability to create new ideas and solutions.

Respondent A shows perseverance by not exiting the software or giving up on conducting the tasks. The respondent tried different ways to solve a problem caused by the software to work out why the software was playing up. Another encounter of respondent A is that their proficiency increased the overall use of the software. The respondent was creative and showed innovation by creating their code during the video footage. Respondent B increases their speed and accuracy when navigating through the tutorial exercises. The software enabled them to work out programming terms by their own cognitive ability to make sure they understood what was happening. The respondent adopted a good understanding of classes, functions and variables through the tutorial and self-research.

They demonstrated creativity and innovation by suggesting new features to create a better work environment and stated, '...get this saving working, it could actually be beneficial.'

The following points work out VisualPro's software satisfaction:-

- **Reusability**: Anything to suggest that the respondent reuses the software may signify the respondents' satisfaction with VisualPro.
- **Feedback**: Feedback is always a great sign of the respondents' satisfaction.

Reviewing the satisfaction from Respondent A proves tricky. Reiterating points like the respondent's reiteration may indicate low satisfaction with the software. This result is shown towards the code generation not yet working as expected, where the respondent was swiftly resetting the software, trying new techniques and was confident using the software. Perhaps, the software proved more useful if desired results were available with less hassle. Respondent B voices his satisfaction with the software with similar concerns. If the code generation were accurate, the respondent would perhaps use it again as they hinted it was an incredible Visual Scripting tool to use in a development environment and less intimidating than traditional programming methods. This respondent, as mentioned previously, pinned the program to the taskbar to show that they would use it again.

11. CONCLUSION

The VisualPro is still in the early stages of the development process. This software iteration can successfully generate readable code. However, there is no doubt about the functionality, UX and User Usability. By looking at figure 11.1 and figure 11.2, the performance suggests that the usability of VisualPro is 'Moderate' usability, and the preference indicates that the usability is 'Easy' usability. The preference suggests that the expectations were that the user usability should end up being easy-to-use, whereas the actual performance states that most users performed moderately well within the environment, where the rest of the users performed better. However, results do show a rough guide of the software's usability, which helps a project in a Iterative development methodology.

A summary of the statistics:- **Performance:**

- *Hard Usability:* Mean Average: 0%.
- *Moderate Usability:* Mean Average: 79.33%.
- *Easy Usability:* Mean Average: 39%.

Preference:

- *Hard Usability:* Mean Average: 0%.
- *Moderate Usability:* Mean Average: 44.5%.
- *Easy Usability:* Mean Average: 57.5%.

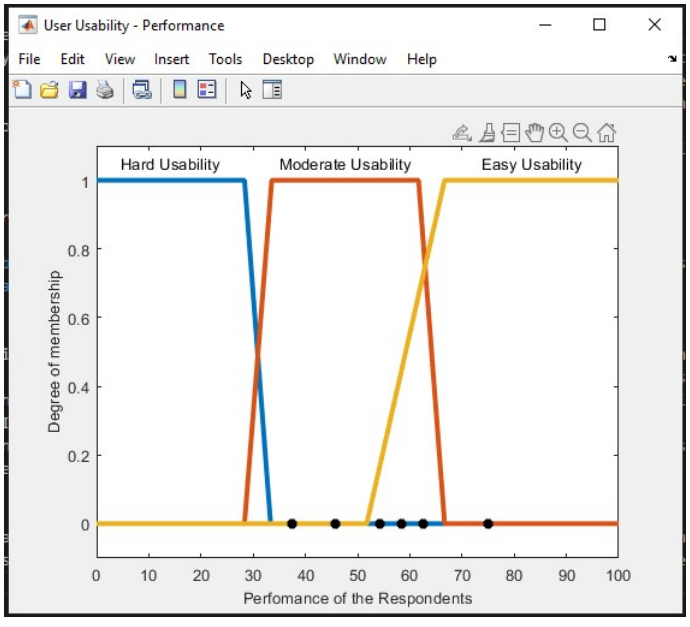

Fig. 11.1. Fuzzy Expert Logic - Performance

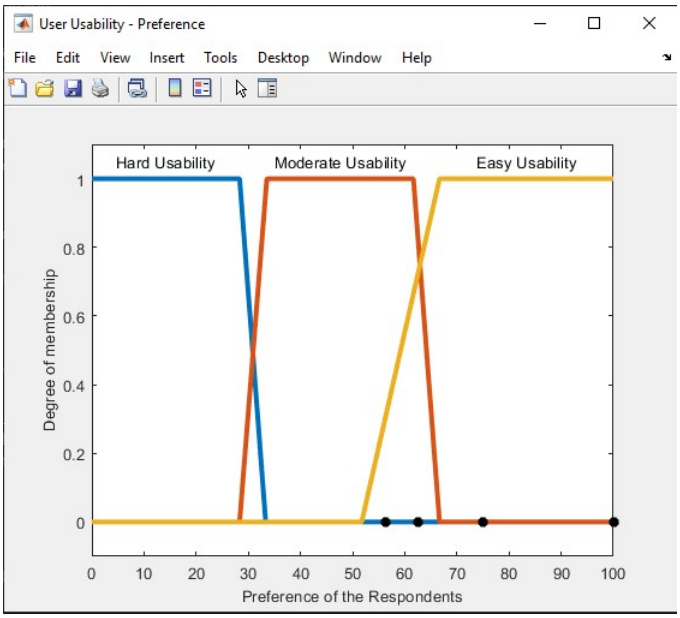

Fig. 11.2. Fuzzy Expert Logic - Preference

12. REFLECTION

13. TERMINOLOGY

List of terminologies used in this document:-

- UI - User Interface.
- UX - User Experience.
- UML - Unified Modeling Languages.
- OOP - Object-Oriented Programming.
- FP - Functional Programming.
- RAISE - Remote Asynchronous Instruction in Science and Engineering.
- DLL - Dynamic-link Library.
- HTML - HyperText Markup Language.
- AI - Artificial Intelligence.
- XML - Extensible Markup Language.
- GUI - Graphical User Interface.

14. APPENDICES

Results - Performance

```
        Respondent 1:
        100% - 3
        75% - 4
        25% - 1
        Mean Average: 78.13%

        Respondent 2:
        100% - 3
        75% - 3
        0% - 2
        Mean Average: 65.63%

        Respondent 3:
        100% - 2
        75 - 4
        0% - 2
        Mean Average: 62.5%

        Respondent 4:
        100% - 1
        75 - 4
        0% - 3
        Mean Average: 50%

        Respondent 5:
        100% - 2
        75% - 1
        25% - 1
        0% - 3

        Mean Average: 37.50%

        Respondent 6:
        75% - 6
        25% - 1

        Mean Average: 59.38%

        Hard Usability:
          I, II, III, IV, V, VI = 0%

        Moderate Usability:
          I - 1
```

```
          II - 1
          III - 1
          IV - 1
          V - 0.76
          VI - 0

          Mean = 79.33%

        Easy Usability
          I - 0
          II - 0
          III - 0.18
          IV - 0.4
          V - 0.76
          VI - 1

          Mean = 39%
```

Results - Preference

```
        Structure:
          Clickable
            Preference: 3 (37.5%)
            Actual: false

          Drag and Drop
            Preference: 5 (62.5%)
            Actual: true

          Drop Down Menu
            Preference: 3 (37.5%)
            Actual: false

          Text Filled
            Preference: 3 (37.5%)
            Actual: false

          Mean Average: 62.5%

        Logic Elements (Variables):
          Clickable
            Preference: 3 (37.5%)
            Actual: false

          Drag and Drop
            Preference: 5 (62.5%)
            Actual: true

```

```
          Drop Down Menu
            Preference: 2 (25%)
            Actual: false

          Text Filled
            Preference: 4 (50%)
            Actual: true

          Mean Average: 56.25%

        Select Language and Save:
          Clickable
            Preference: 4 (50%)
            Actual: false

          Drag and Drop
            Preference: 1 (12.5%)
            Actual: false

          Drop Down Menu
            Preference: 6 (75%)
            Actual: true

          Text Filled
            Preference: 2 (25%)
            Actual: false

          Mean Average: 75%

        Sub Elements:
          Clickable
            Preference: 4 (50%)
            Actual: false

          Drag and Drop
            Preference: 4 (50%)
            Actual: true

          Drop Down Menu
            Preference: 3 (37.5%)
            Actual: false

          Text Filled
            Preference: 5 (62.5%)
            Actual: true

          Mean Average: 56.25


        Operating System (PC):
          Windows OS:
```

```
            Preference: 8 (100%)
            Actual: true
          Linux:
            Preference: 4 (50%)
            Actual: false

          ChromeOS/Android OS:
            Preference: 3 (37.5%)
            Actual: false

          macOS/iOS:
            Preference: 2 (25%)
            Actual: false

          Mean Average: 100%

        Operating System (Web Application):
          Windows OS:
            Preference: 8 (100%)
            Actual: false
          Linux:
            Preference: 4 (50%)
            Actual: false
          ChromeOS/Android OS:
            Preference: 2 (25%)
            Actual: false
          macOS/iOS:
            Preference: 1 (12.5%)
            Actual: false

          Mean Average: 0%

        Operating System (Mobile Devices):
          Windows OS:
            Preference: 1 (12.5%)
            Actual: false
          Linux:
            Preference: 1 (12.5%)
            Actual: false
          ChromeOS/Android OS:
            Preference: 6 (75%)
            Actual: false
          macOS/iOS:
            Preference: 4 (50%)
            Actual: false

          Mean Average: 0%


        Overall Mean Average: 70%

```

```
        Hard Usability:
          I, II, III, IV = 0%

        Moderate Usability:
          I - 1
          II - 0.78
          III - 0
          IV - 0

          Mean = 44.5%

        Easy Usability
          I - 0.3
          II - 0
          III - 1
          IV - 1

          Mean = 57.5%
```

ACKNOWLEDGEMENT

Appreciation goes out to the survey participants, which helped better understand how VisualPro may impact consumers in the future. Having the support from these participants helped advance the UI and UX progress of VisualPro.

BIBLIOGRAPHY

[1] B. Vogel-Heuser, M. Obermeier, S. Braun, K. Sommer, F. Jobst, and K. Schweizer, "Evaluation of a UML-Based Versus an IEC 61131-3-Based Software Engineering Approach for Teaching PLC Programming," *IEEE Transactions on Education*, vol. 56, no. 3, pp. 329–335, 2013.

[2] K. Kuspert, J. Teuhola, and L. Wegner, "Design issues and first experiences with a visual database editor for the extended NF/sup 2/-data model," in *Twenty-Third Annual Hawaii International Conference on System Sciences*, vol. 2, 1990, pp. 308–317 vol.2.

[3] Chisnell Dana and Rubin Jeffrey, *Handbook of usability testing how to plan, design, and conduct effective tests*, 2nd ed. Indianapolis, IN: Wiley Pub, 2008.

[4] D. Zuehlke and N. Thiels, "Useware engineering: a methodology for the development of user-friendly interfaces," *Library Hi Tech*, vol. 26, no. 1, pp. 126–140, 2008, iSBN: 07378831. [Online]. Available: https://www-proquest-com.ezproxy.uwtsd.ac.uk/scholarly-journals/useware-engineering-methodology-development-user/docview/200606343/se-2?accountid=130472

[5] M. Hentati, L. Ben Ammar, A. Trabelsi, and A. Mahfoudhi, "A fuzzy-logic system for the user interface usability measurement," in *2016 17th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*, May 2016, pp. 133–138.

[6] Dr. Caitlin Sadowski and Dr. Thomas Zimmerman, Eds., *Rethinking Productivity in Software Engineering*, 1st ed. Apress, 2019. [Online]. Available: https://doi.org/10.1007/978-1-4842-4221-6

[7] Unreal Engine, "Introduction to Blueprints." [Online]. Available: https://docs.unrealengine.com/4.27/en-US/ProgrammingAndScripting/Blueprints/GettingStarted/

[8] Tileston Donna Walker, *10 best teaching practices how brain research and learning styles define teaching competencies*, 3rd ed. Thousand Oaks, Calif. ;: Corwin, 2011.

[9] T. Cleaver, "Interactive Web-based tutorials for engineering education," in *Proceedings IEEE Southeastcon'99. Technology on the Brink of 2000 (Cat. No.99CH36300)*, Mar. 1999, pp. 126–127.

[10] R. Krull, "Three levels of guidance in technical tutorials," in *Communication Proceedings of IPCC 97*, Oct. 1997, pp. 441–448.

[11] Pooley R J and Stevens Perdita, *Using UML: software engineering with objects and components*, second edition. ed., ser. Addison-Wesley Object Technology Series. Harlow, England: Pearson Education Limited, 2006.

[12] Unity Technologies, "Unity engine visual scripting | Game development software without coding | Unity Bolt | Unity." [Online]. Available: https://unity.com/products/unity-visual-scripting

[13] Mojang, "Minecraft - Getting Started with Redstone." [Online]. Available: https://help.minecraft.net/hc/en-us/articles/360045950932-Minecraft-Getting-Started-with-Redstone-

[14] C. Klammer and A. Kern, "Writing unit tests: It's now or never!" in *2015 IEEE Eighth International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, Apr. 2015, pp. 1–4.

[15] Z. Chaczko, R. Braun, L. Carrion, and J. Dagher, "Design of unit testing using xUnit.net," in *2014 Information Technology Based Higher Education and Training (ITHET)*, 2014, pp. 1–9.

[16] Krug Steve, *Don't make me think, revisited: a common sense approach to web usability*, third edition ed. Berkeley: New Riders, 2014.

[17] J. Nielsen and J. Levy, "Measuring usability: preference vs. performance," *Communications of the ACM*, vol. 37, no. 4, pp. 66–75, 1994, place: New York, NY Publisher: ACM.

[18] E. Patch, "VisualPro (Visual Scripting Pad)," Jan. 2022, original-date: 2021-09-24T08:51:54Z. [Online]. Available: https://github.com/ShinkuKira21/VisualPro-FinalProject

[19] "Code Faster with AI Code Completions | Tabnine." [Online]. Available: https://www.tabnine.com/

[20] L. R. Vijayasarathy and C. W. Butler, "Choice of Software Development Methodologies: Do Organizational, Project, and Team Characteristics Matter?" *IEEE Software*, vol. 33, no. 5, pp. 86–94, Sep. 2016.

[21] A. M. Winn and T. J. Smedley, "Multimedia Workshop: Exploring the Benefits of a Visual Scripting Language." IEEE Computer Society, Sep. 1998, pp. 280–280. [Online]. Available: https://www.computer.org/csdl/proceedings-article/vl/1998/87120280/12OmNBl6EH3

[22] E. T. Ray, *Learning XML*, 2nd ed. Sebastopol, California: O'REILLY.

[23] G. K. Behara, *Why Python is Popular for Machine Learning Implementations?* New Dehli: Athena Information Solutions Pvt. Ltd.

[24] "Markup Languages: Theory & Practice," *Markup Languages: Theory & Practice*, vol. 1, no. 4, pp. 46–46, 1999, publisher: MIT Press. [Online]. Available: https://ezproxy.uwtsd.ac.uk/login?url=https://search.ebscohost.com/login.aspx?direct=true&db=aph&AN=7317014&site=ehost-live

[25] Microsoft, "Compare Project Management Solutions and Costs | Microsoft Project." [Online]. Available: https://www.microsoft.com/en-gb/microsoft-365/project/compare-microsoft-project-management-software

[26] G. Costa and R. Ortale, "Machine learning techniques for XML (co-)clustering by structure-constrained phrases," *Information Retrieval*, vol. 21, no. 1, pp. 24–55, Feb. 2018, num Pages: 24-55 Place: Dordrecht, Netherlands Publisher: Springer Nature B.V. [Online]. Available: http://www.proquest.com/docview/2002401888/abstract/100992F450AA4EEFPQ/1

[27] Microsoft, "Exporting from a DLL Using __declspec(dllexport)," May 2019. [Online]. Available: https://docs.microsoft.com/en-us/cpp/build/exporting-from-a-dll-using-declspec-dllexport

[28] ——, "DllImportAttribute Class (System.Runtime.InteropServices)." [Online]. Available: https://docs.microsoft.com/en-us/dotnet/api/system.runtime.interopservices.dllimportattribute

[29] ——, "What is .NET Framework? A software development framework." [Online]. Available: https://dotnet.microsoft.com/learn/dotnet/what-is-dotnet-framework

[30] GitHub, "GitHub: Where the world builds software." [Online]. Available: https://github.com/

[31] Git, "GitSCM." [Online]. Available: https://git-scm.com/

[32] Microsoft, "Visual Studio: IDE and Code Editor for Software Developers and Teams." [Online]. Available: https://visualstudio.microsoft.com/

[33] "Full details and actions for Creating the productive workplace." [Online]. Available: https://www-vlebooks-com.ezproxy.uwtsd.ac.uk/Vleweb/Product/Index/62495?page=0

[34] D. Clements-Croome, Ed., *Creating the Productive Workplace*, 2nd ed. 2 Park Square, Milton Park, Abingdon, Oxon OX14 4RN: TAYLOR & FRANCIS, Aug. 2006.

[35] M. Kalicinski, "RapidXml," 2009. [Online]. Available: http://rapidxml.sourceforge.net/

[36] Epic Games, "Unreal Engine 5." [Online]. Available: https://www.unrealengine.com/en-US/unreal-engine-5

[37] E. Arisholm, L. C. Briand, S. E. Hove, and Y. Labiche, "The Impact of UML Documentation on Software Maintenance: An Experimental Evaluation," *IEEE Transactions on Software Engineering*, vol. 32, no. 6, pp. 365–381, Jun. 2006. [Online]. Available: https://www.proquest.com/docview/195572945/abstract/75B2DD5585C6437DPQ/1

[38] Wiklund Michael E, *Usability in practice: how companies develop user-friendly products*. Boston ;: AP Professional, 1994, book Title: Usability in practice : how companies develop user-friendly products.

[39] Albert Bill and Tullis Tom, *Measuring the user experience: collecting, analyzing, and presenting usability metrics*, ser. The Morgan Kaufmann series in interactive technologies. Amsterdam ;: Morgan Kaufmann, 2008, book Title: Measuring the user experience : collecting, analyzing, and presenting usability metrics.

[40] "Understanding new and emerging user values," *Telesis*, no. 97, p. 14, Dec. 1993, iSBN: 00402710. [Online]. Available: https://www-proquest-com.ezproxy.uwtsd.ac.uk/trade-journals/understanding-new-emerging-user-values/docview/222355229/se-2?accountid=130472

[41] P. Capek, E. Kral, and R. Senkerik, "Advanced Testing Tool for .NET Applications," in *2017 International Conference on Computational Science and Computational Intelligence (CSCI)*, 2017, pp. 1799–1800.

[42] L. Mukkavilli, "Smart Unit Testing Framework," in *2012 IEEE 23rd International Symposium on Software Reliability Engineering Workshops*, 2012, pp. 70–79.

[43] J. Cho, "Getting Into the Field," in *Evaluating Qualitative Research*. New York: Oxford University Press, 2017. [Online]. Available: https://oxford.universitypressscholarship.com/10.1093/oso/9780199330010.001.0001/oso-9780199330010-chapter-1

[44] I. Grout and A. K. B. A'Ain, "Adapting an on-line tutorial tool with web analytics to incorporate analysis of tutorial use," in *2012 15th International Conference on Interactive Collaborative Learning (ICL)*, Sep. 2012, pp. 1–7.

[45] P. Gopal, N. Varkey, and K. M. Moudgalya, "Visual Maps for Collaborative Spoken Tutorial Development," in *2012 IEEE Fourth International Conference on Technology for Education*, Jul. 2012, pp. 253–254.

[46] C. Troussas, K. Chrysafiadi, and M. Virvou, "Personalized tutoring through a stereotype student model incorporating a hybrid learning style instrument," *Education & Information Technologies*, vol. 26, no. 2, pp. 2295–2307, Mar. 2021. [Online]. Available: https://ezproxy.uwtsd.ac.uk/login?url=https://search.ebscohost.com/login.aspx?direct=true&db=bri&AN=149397930&site=ehost-live

[47] Z. Luo, "Using eye-tracking technology to identify learning styles: Behaviour patterns and identification accuracy," *Education & Information Technologies*, vol. 26, no. 4, pp. 4457–4485, Jul. 2021. [Online]. Available: https://ezproxy.uwtsd.ac.uk/login?url=https://search.ebscohost.com/login.aspx?direct=true&db=bri&AN=151490161&site=ehost-live

[48] Lo Mun Ling, Pong Wing Yan, and Chik Pui Man Pakey, *For each and everyone catering for individual differences through learning studies*. Hong Kong: University Press, 2005.

[49] Pritchard Alan, *Ways of learning: learning theories for the classroom*, fourth edition. ed. Abingdon, Oxon ;: Routledge, 2018.

[50] L. Shao, F. Zhu, and X. Li, "Transfer Learning for Visual Categorization: A Survey," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 26, no. 5, pp. 1019–1034, May 2015.

[51] R. Crepon, "Application of design research methodology to a contex-sensitive study in engineering education," in *2014 IEEE Frontiers in Education Conference (FIE) Proceedings*, Oct. 2014, pp. 1–4, iSSN: 2377-634X.

[52] E. T. Mushashu and J. S. Mtebe, "Investigating Software Development Methodologies and Practices in Software Industry in Tanzania," in *2019 IST-Africa Week Conference (IST-Africa)*, May 2019, pp. 1–11, iSSN: 2576-8581.

[53] S. Pontie, A. Bourge, A. Prost-Boucle, P. Maistri, O. Muller, R. Leveugle, and F. Rousseau, "HLS-Based Methodology for Fast Iterative Development Applied to Elliptic Curve Arithmetic," in *2016 Euromicro Conference on Digital System Design (DSD)*, Aug. 2016, pp. 511–518.

[54] T. Kister, "Improving the information development process: A refined iterative development model," vol. 63, pp. 186–211, Aug. 2016.

[55] O. Galkina and V. Yachenko, "Application of iterative software development methodologies for reducing service quality gaps," in *Proceedings of the 2014 IEEE NW Russia Young Researchers in Electrical and Electronic Engineering Conference*, Feb. 2014, pp. 36–37.

[56] M. Javanmard and M. Alian, "Comparison between Agile and Traditional software development methodologies," *Cumhuriyet Üniversitesi Fen Edebiyat Fakültesi Fen Bilimleri Dergisi*, vol. 36, no. 3, pp. 1386–1394, May 2015. [Online]. Available: https://dergipark.org.tr/en/pub/cumuscij/issue/45132/564438

[57] G. W. Sasmito, L. O. M. Zulfiqar, and M. Nishom, "Usability Testing based on System Usability Scale and Net Promoter Score," in *2019 International Seminar on Research of Information Technology and Intelligent Systems (ISRITI)*, Dec. 2019, pp. 540–545.

[58] F. Dias and A. C. R. Paiva, "Pattern-Based Usability Testing," in *2017 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, Mar. 2017, pp. 366–371.

[59] A. L. Iniguez-Carrillo, L. S. Gaytan-Lugo, M. A. Garcia-Ruiz, and R. Maciel-Arellano, "Usability Questionnaires to Evaluate Voice User Interfaces," *IEEE Latin America Transactions*, vol. 19, no. 9, pp. 1468–1477, Sep. 2021.

[60] M. R. H. Iman and A. Rasoolzadegan, "Quantitative evaluation of software usability with a fuzzy expert system," in *2015 5th International Conference on Computer and Knowledge Engineering (ICCKE)*, Oct. 2015, pp. 325–330.