VisualPro

1st Given Edward Patch
Software Engineer Student (of BSc Year 3)
Independent Project
University of Wales Trinity St. Davids (of Mike Dacey)
Swansea, Wales
Student ID: 1801492

CONTENTS

***Abstract*—**
***Index Terms*—Visual Programming, Visual Scripting, Planner**

I. INTRODUCTION

II. LITERATURE REVIEW

The Literature Review section covers 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 in ahead of time.

A. Pre-Planning

1) HTML Design: The figure 1 shows a HyperText Markup Language (HTML) Design of how the User Interface 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. 1. HTML Design - Found at: Original Image

2) Notebook Page 1-2: Figure 2 contains numerous ideas. Page 1 contains a heading, 'Personal Assistant', which indicates the author wanted to make a personal assistant at first. After planning this, the author wanted 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 (ref here).

The second page of this figure, shows a few different examples of IF statement syntax from languages 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.

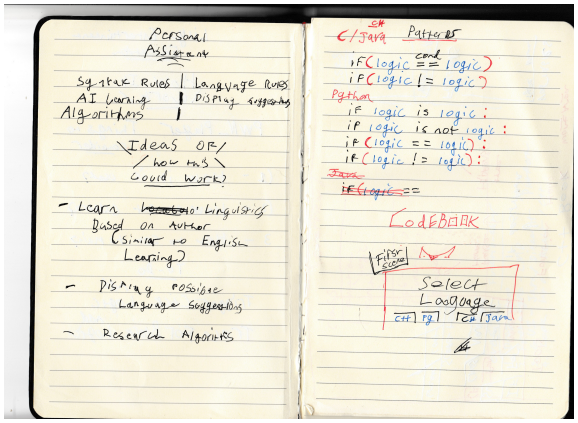

Fig. 2. Notebook Plan Page 1-2 - Found at: Original Image

3) Notebook Page 2-3: The Figure 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.

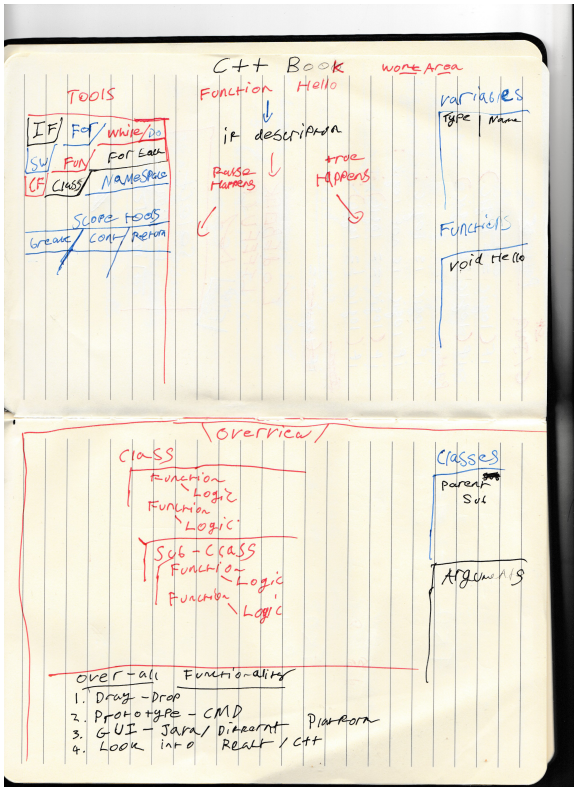

Fig. 3. Notebook Plan Page 3-4 - Found at: Original Image

4) Notebook Page 5-6: The Figure 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.

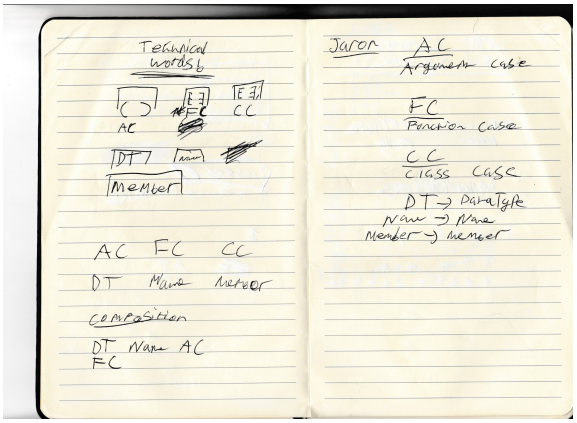

Fig. 4. Notebook Plan Page 5-6 - Found at: Original Image

5) Notebook Page 7-8: The figure 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.

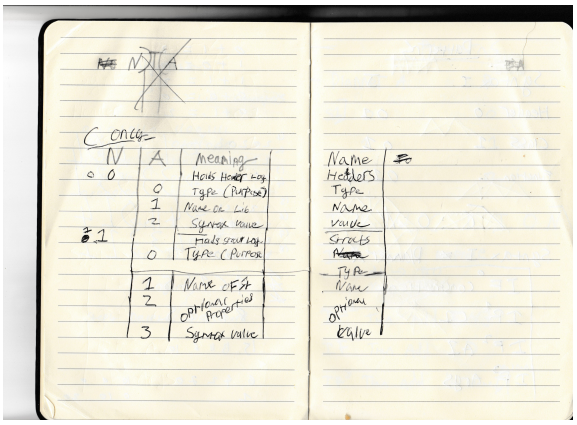

Fig. 5. Notebook Plan Page 7-8 - Found at: Original Image

6) Notebook Page 9-10: The figure 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.

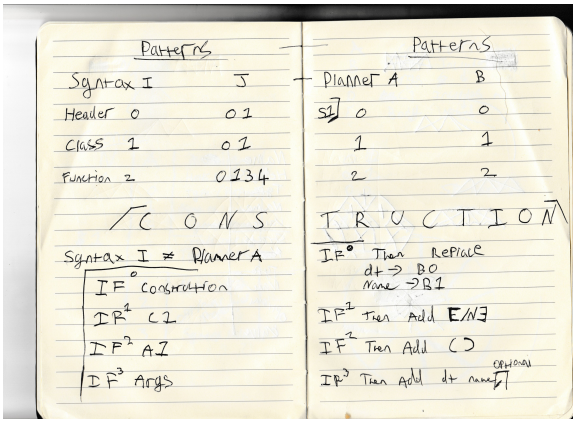

Fig. 6. Notebook Plan Page 9-10'' - Found at: Original Image

7) Notebook Page 11: The figure 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 programmatical sense. This again helped when it comes to the prototypes further in this document.

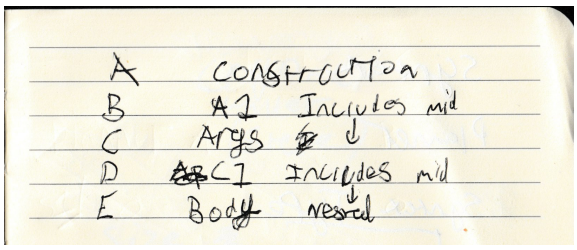

Fig. 7. Notebook Plan Page 11'' - Found at: Original Image

8) Notebook Page 12-13: A predicted problem is as follows:

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

Answer:
The planner would hold it's parent and sub identity, and the Triangle (30 Code) is used to tell the software where to put the child. The triangle 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 problem before is that in theory, it would be hard to tell the program to remember when and where to close tags if it got too deep.

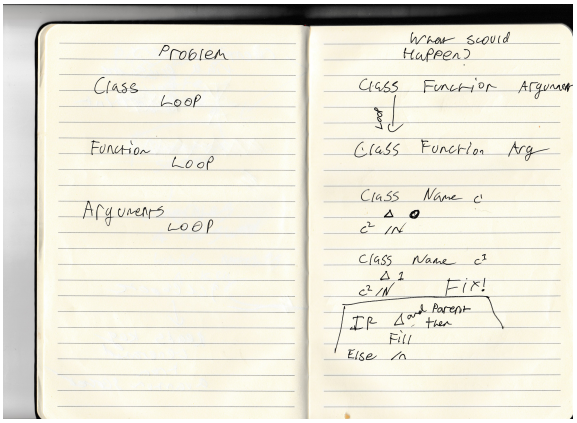

Fig. 8. Notebook Plan Page 12-13'' - Found at: Original Image

B. Surveying

III. Prototypes

A. Design and Implementation

B. Test Plan

C. Recorded Problems

D. Unit Testing

IV. Terminology

List of terminologies used in this document:-

- HTML - HyperText Markup Language.
- AI - Artificial Intelligence.
- XML - Extensible Markup Language.

V. Appendices

VI. Reference List