PDP - Pen Driven Programming

Jonathan Frye
The University of Edinburgh
School of Informatics
EH8 9LE
Jonathan@frye.org.uk

Björn Franke
The University of Edinburgh
School of Informatics
EH8 9LE
bfranke@inf.ed.ac.uk

ABSTRACT
Programming is an activity centred primarily around the keyboard which is not necessarily the optimal input device for all users. Little research has taken place into alternative input devices for programming despite huge advances in handwriting and voice recognition for natural language. This project explored using a pen as the primary input device for programming. A variety of different methods for using the pen were designed, developed and evaluated. Existing variable and method declarations were used in the handwriting recognition to improve its accuracy. Additionally code generation techniques were explored to minimize the volume of writing required. These features were then integrated into Microsoft Visual Studio 2005, a commercial IDE, to enable the evaluation of a pen driven environment complete with all the features expected of a modern day IDE.

Categories and Subject Descriptors
D.2.6 [**Software Engineering**]: Programming Environments

General Terms
Human Factors

Keywords
Pen Driven Programming, Tablet PC, RSI

1. INTRODUCTION
Users who suffer from RSI find that extended use of a keyboard and mouse can be extremely painful [1]. Those users who do not have full mobility in a hand may find that typing is slower than usual. A more natural input device is the pen which is used frequently in everyday life without the problems associated with a keyboard. To date there has been a lot of work on handwriting and speech recognition in relation to spoken language but very little research into its application to programming. The goal of this project was to focus on the use of a pen for programming using an existing Integrated Development Environment **(**IDE**)**.

Previous work has been undertaken to research alternative interfaces to the PC. Pendragon [3] is a hardware and software project run at Georgia Institute of Technology which allows for rapid prototyping of more natural hardware and software interfaces. The project is aimed at people who are unable to use a keyboard and seeks to find more comfortable input methods which can come close to the speed and accuracy of a keyboard.

The approach is to split up programming into three tasks: text entry, text modification and navigation. The report points out that programming does not fit naturally into a system written for recognising English due the punctuation and spacing necessary for most languages. However it is noted that using a variety of techniques such as word completion the speed of entering text could be increased. It is also suggested that a programming language's strict and well defined grammar may be an advantage to a handwriting recogniser written specifically for programming.

The hardware used for the project is vastly different to that of a standard workstation. The monitor has been replaced by projecting the image directly onto the desktop. The mouse is replaced by a foot pedal and the keyboard by a pen tablet.

The team use this environment to prototype innovative interfaces. One such system is a soft keyboard named Cirrin [4] which presents the characters in a circle. A word is created by moving the pen from one character to the next. It is reported that this solution is sufficiently fast to be usable for writing out email on a daily basis and speeds of 20wpm have been achieved by a trained and practiced user.

Numerous systems exist today for performing handwriting recognition on written English. At a basic level such a system can be used for programming but the differences between English and source code are sufficient to make this a slow and unreliable method. The main problem is that handwriting recognition systems are built to take advantage of the rules of the English language [6]. Simple assumptions such as a space always follows a full stop do not hold true for object oriented languages. Variable names are frequently created from concatenated words. This poses a problem for the handwriting recognition as it expects spaces to appear between words contained within its dictionary.

Pen Driven Programming is a project to tackle these shortcomings and provide pen based features as part of a fully functional IDE. Microsoft Visual Studio 2005 was selected as the target IDE as it is widely used commercially and provides an extensibility platform. The system was developed to be used on a Tablet PC; a Lenovo X61 was used for the development and testing of this project. The implementation targeted developing in the C# language but many of the techniques generalise to a wide variety of other languages.

2. DESIGN AND IMPLEMENTATION
The user interface consists of two separate windows; the first is predominantly for writing out source code while the second is used for code generation and navigation. Both windows use the pen as the sole input device and make extensive use of the existing source code and its structure. The Microsoft Tablet PC API [5] was used to provide the handwriting recognition.

2.1 Handwriting Area

The handwriting area (see Figure 1) is a window providing the user with a variety of methods to enter source code. The first of these methods is a large handwriting area where code can be written out a line at a time. As the user writes the handwriting strokes are analyzed in the background and three possible suggestions displayed below. To enter a suggestion into the source code window the user simply has to tap the suggestion with the pen.

Initial testing showed that as the entered text became longer the probability of obtaining a 100% correct suggestion dropped. To combat this the entered text was split up into individual tokens, based on spaces and '.' then suggestions were offered for each sub-word using a drop down. This allowed a far greater number of combinatorial suggestions within a constrained space.

The project made use of the Microsoft Tablet PC API to perform the recognition. However this recognizer was developed for written English, not for C#. This posed several problems such as recognizing '(' as 'C' and refusal to accept obscure variable names. A variety of approaches were taken to overcome these limitations. The code in the source code window was parsed and a list of declared variables and functions obtained. This list was then fed, along with the basics of the C# syntax, to the recognition system as a custom dictionary. This method greatly improved the recognition accuracy of identifiers such as 'bool' or previously declared variable names.

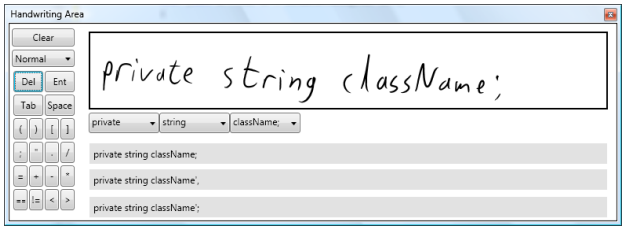

Figure 1. Handwriting Area

Declaring variable names for the first time remained problematic if they were not made up from the English language. This was tackled by providing a special variable entry mode where the handwriting input area is split up into boxes for individual characters which are then recognized individually. This method is substantially slower for the user as each character must be entered individually but it only needs to be done for the first declaration. Subsequent uses of the variable are more accurately detected as the word now exists in the custom dictionary fed to the recognizer.

Punctuation is used much more widely in source code than it is written English, brackets in particular were problematic for the handwriting recognition system. To speed up program entry a small on screen keyboard was created out of buttons which contained the 16 most frequently used punctuation symbols along with Delete, Space, Enter and Tab (see Figure 2).

Figure 2. Onscreen Keyboard

2.2 Code Structure Tree

The Code Structure Tree (see Figure 3) displays the structure of the currently open source file as a tree. The user can use this tree in a variety of ways including navigating to declarations and code generation. The tree structure is generated from the source code and displays the classes, methods and variables currently declared. Variables declared within a method are shown as children of the method to produce the hierarchical structure. Double clicking a node jumps to the corresponding declaration in the source code to allow for easy navigation without scrolling.

Writing out code using the pen was found to be slow compared with using the keyboard. While the goal of the project was not to produce a faster interface for an able programmer, it was desirable to provide an interface which is as fast as possible. Code generation was used to speed up program entry once variables have been declared. The code generation is accessed by right clicking a declared method or variable in the tree and selecting a construct from the menu.

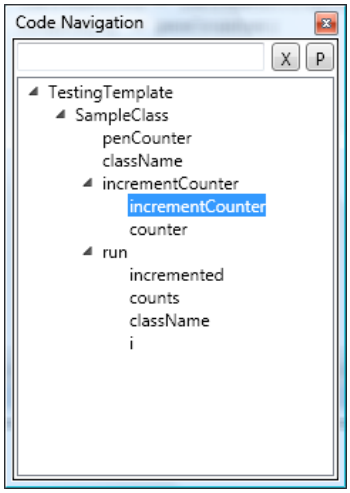

Figure 3. Code Structure Tree

The code generation is sensitive to the data type of the variable being used. For example selecting the "For Loop" construct from an Integer *var* would generate code which loops *var* times:

```
for (int i = 0; i < var; i++) {

}
```

By contrast selecting the "For Loop" construct from an array *var* would iterate around each element in the array:

```
for (int i = 0; i < var.length(); i++) {

}
```

The Code Structure Tree was also made searchable to rapidly locate variables. Usually text is entered using a dedicated handwriting area which uses up screen space unnecessarily. To search the tree the user simply writes their search term directly on top of the treeview (see Figure 4). This was achieved by overlaying the treeview with a transparent handwriting input area and then recognizing the writing as normal. This system provided the user with an intuitive method of searching while preserving valuable screen estate.

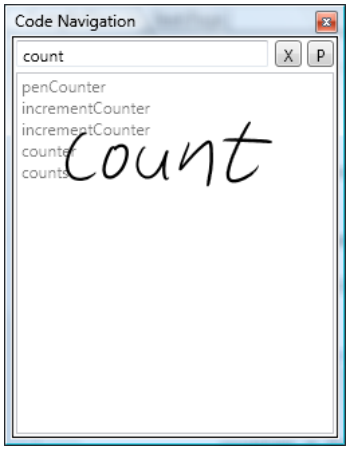

Figure 4. Treeview Search

2.3 IDE Integration

The Handwriting Area and the Code Structure Tree were developed as Visual Studio 2005 Integration packages. This allowed them to be opened as tool windows from within the IDE and then dragged around, resized or docked by the user. Both use the currently active code window to provide a list of declarations to produce more accurate handwriting recognition results. The full system can be seen in Figure 5 with the Handwriting Area at the bottom of the screen and the Code Structure Tree at the left hand side.

3. EVALUATION

A small scale usability test was undertaken at the end of the project by a cross-section of users. Each user was given a brief introduction to the modified IDE and then asked to copy out a segment of code. The code was specifically written with the developed features in mind so it provided ample scope to make use of the new interface. The users were then asked to answer 18 questions about individual features and the system as a whole. Two of the users who participated suffered mildly from RSI and one was left handed.

The results relating to the handwriting area were very positive. Users were consistently impressed with the handwriting recognition accuracy and the use of alternate recognisers. It was clear from the results that the Onscreen Keyboard was a necessary addition with all users finding it to be useful. A comparison with the keyboard was less good with most users being unconvinced that the pen was easier to use than the keyboard. It should be noted that most of the participants had no issues using a keyboard at all and two suffered mildly from RSI but not so much as to prevent keyboard use.

The feedback on the Code Structure Tree was positive with most users finding it easy to use and generally useful. Locating variables and searching the tree were features which the users viewed to be only moderately useful. This may be in part due to the fact that the class they were using was very small so locating variables was not particularly difficult anyway and there were too few options to make search necessary. Observations showed that users tended to write out variables by hand rather than insert them from the tree. This may have been because inserting was a rather cumbersome action but it could also have been influenced by the fact they were copying code rather than trying to think what variable it was that they wanted. Most users found writing directly on the treeview to search a natural method once they were shown how to do it but a couple were unconvinced by this method. Almost all users were in agreement that the code generation was substantially easier than writing out code by hand.

There was a large range of opinions as to how easy the system was to use ranging across almost the entire scale. The average opinion was that it is fairly easy to use though there is some scope for improvement. There was broad agreement that the system was slower than using the keyboard. It was never a design goal to produce a faster interface than the keyboard as the system is aimed at people who are unable to use a keyboard.

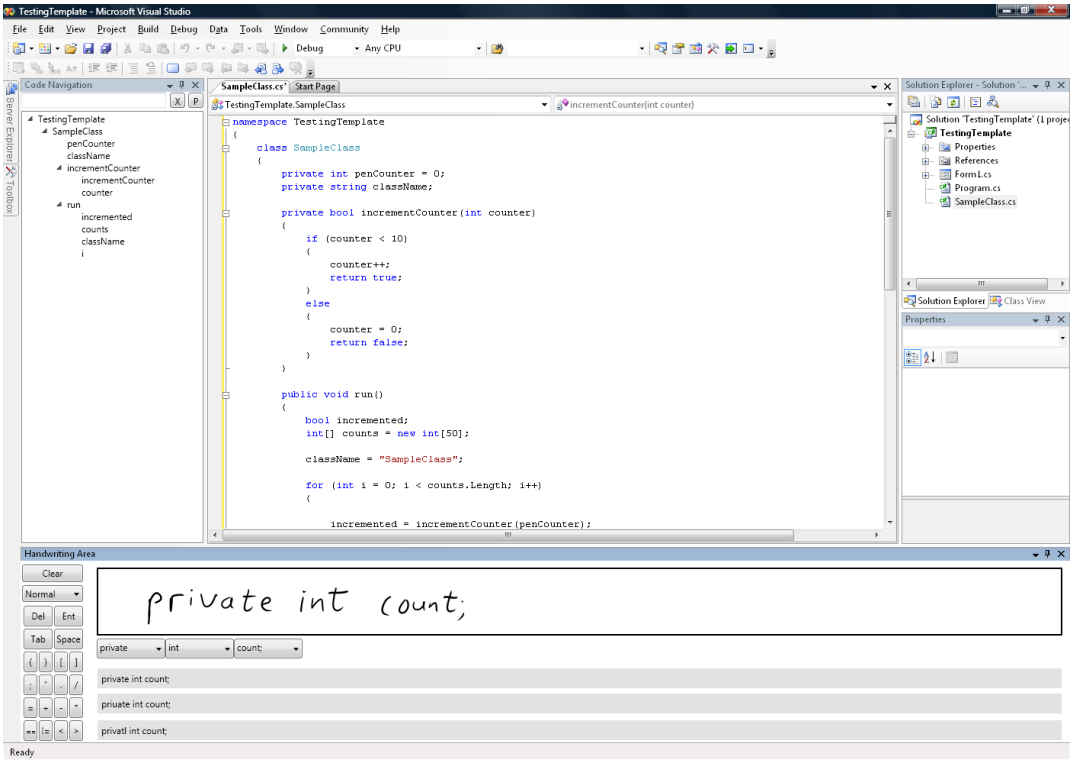

Figure 5. Full IDE Integration

This is an interesting comparison however as it is desirable for such a system to be able to rival other methods in terms of speed. There was no agreement about the comfort of the pen as compared to the keyboard; the responses covered the whole range with an average almost exactly in the middle. Interestingly the two users with RSI reported dramatically different results with one finding the pen to be very comfortable and the other found it to be even worse than a keyboard. No conclusion can be drawn from this other than PDP may be suitable for some users but not others. Most users would still like to have a keyboard to be available while using the system for persistently difficult words; these tended to be the users who had encountered particular difficulty with the handwriting recognition during the test. Only the left handed user had difficulty with their hand coving part of the interface. This was expected as the system was developed for right handed users but it shows that the interface will need to be adapted to accommodate left handed users.

Whilst the project was developed specifically with a tablet PC in mind it had some shortcomings. In particular the 12.1" screen was very small when compared to what most software developers use today. Typically developers use high powered desktop computers with large monitors so a method of integrating PDP with such environments was evaluated. The tablet PC was connected to an external monitor, keyboard and mouse. This provided a normal desktop type setup with the addition of a touch sensitive second monitor.

Visual Studio was setup so as to span both monitors, the large external monitor displayed Visual Studio as usual while the tablet PC screen just displayed the Handwriting and Code Structure areas. This effectively turned the tablet PC into an alternative keyboard and allowed the external monitor to display an unaltered IDE. The addition of a keyboard meant that particularly tricky words could be typed rather than persevering with the recognition. This system performed well and would be a workable solution at least as good as just using the tablet PC. The costs of a monitor, keyboard and mouse are minimal in comparison to the tablet PC. If this avenue were to be pursued a lot more screen space would be available for additional pen features.

4. CONCLUSIONS

This project successfully developed a system for programming with a modern IDE using a pen as the primary input device. A variety of features such as parsing the existing code for variables and context sensitive code generation were implemented to improve the accuracy and speed of the system. The results of the project show that using a written English handwriting recognition system is not sufficient. Knowledge of the programming language syntax and variable names is required achieve a high accuracy rate. Punctuation detection proved to be problematic initially but it was solved with the addition of a simple on-screen keyboard.

Code generation was found to speed up development considerably as writing out code by hand is slow. This project only implemented a handful of code generation templates but this figure could easily be increased to several hundred without any difficulty.

Since the project was implemented using Visual Studio Integration Packages it can easily be used alongside any other tools which also integrate into Visual Studio. This provides great scope for using a variety of different inputs methods rather than being restricted to one. Allowing multiple modes of input offers the possibility to change between them in order to reduce the strain on any single part of the body. There is great potential in a system that combines both voice and pen input as they are activities which can occur simultaneously.

We identified that there are several areas where using voice input would be easier than the pen. The Program Structure Tree could be augmented with voice recognition instead of using a system of cascading menus. Having selected the required variable in the tree with the pen the user could dictate a command such as "create for loop" which would produce a For Loop around the selected variable. This is quicker and more intuitive than navigating a system of menus, especially if the number of possible code generation templates were to be expanded.

Combining the advantages of the pen and voice would overcome some of their respective disadvantages to create an environment more comparable to the traditional keyboard and mouse combination. The results of the study into Keyboardless Visual Programming [2] showed that such environments can be highly successful.

5. REFERENCES

[1] Arnold, S. C., Mark, L., and Goldthwaite, J. Programming by voice, vocalprogramming. In *Assets '00: Proceedings of the fourth international ACM conference on Assistive technologies* (New York, NY, USA, 2000). ACM, 149-155

[2] Leopold, J. L., and Ambler, A. L. Keyboardless visual programming using voice, handwriting, and gesture. In *VL '97: Proceedings of the 1997 IEEE Symposium on Visual Languages (VL '97)* (Washington, DC, USA, 1997). IEEE Computer Society, 28.

[3] Manko, J., Abowd, G., and Goldthwaite, J. Pendragon. http://www-static.cc.gatech.edu/fce/pendragon/

[4] Manko, J. and Abowd, G. Cirrin: A word-level unistroke keyboard for pen input. In *Proceedings of UIST* (San Francisco, California, November 1998). ACM Press, 213-214

[5] Microsoft. Tablet PC Development Guide. http://msdn.microsoft.com/en-us/library/ms704849(VS.85).aspx

[6] West, J. V. Using Tablet PC: Handwriting Recognition 101. http://www.microsoft.com/windowsxp/using/tabletpc/getstarted/vanwest_03may28hanrec.mspx