Software to Sketch Interface Designs

Beryl Plimmer

Department of Computer Science,
University of Auckland
Private Bag 92019, Auckland,
New Zealand

b.plimmer@auckland.ac.nz

Mark Apperley

Department of Computer Science
University of Waikato
Private Bag 3105, Hamilton,
New Zealand

m.apperley@cs.waikato.ac.nz

Abstract: This paper describes the development and evaluation of an electronic sketch environment for interface design. The tool provides a pen-based interface on an electronic whiteboard for designing Visual Basic forms, it is tightly integrated into the Visual Basic IDE. Our evaluation showed that this type of environment is likely to be of benefit to novice programmers as it provides an enticing shared workspace for small groups and encourages checking and revision.

Keywords: Sketching, public space interfaces, informal design, novice programmers

1 Introduction

Learning to program is difficult, in simplistic terms the novice programmer must learn the programming language syntax, and how to decompose problems and recompose them as an algorithm. User interfaces are a significant part of problem solutions and an obvious focus to explore problem requirements.

Designers from a wide range of disciplines initially hand-sketch their ideas because informal tools offer the freedom to work with partly formed or ambiguous ideas. Generally once a design is well defined it is transferred to a computer-based tool which offers other advantages such as easy editing and distribution. Student programmers tend not to hand-sketch their interfaces because they see it as a waste of time.

We have developed an informal design tool (Freeform) that is tightly integrated into a programming IDE. It uses pen input on a digital whiteboard to imitate the informality of a low-fidelity tool, but at the same time offers the functional support expected in a computer environment. The computer environment also makes it possible to simulate the sketched interface in operation.

Our evaluation showed that students using Freeform designed more appropriate interfaces because they were more involved in the design process. They also enjoyed the experience and developed a more positive attitude to sketching.

2 Background

This project brings together ideas from graphic design, program design, and educational psychology. The user interface is an important part of any program and offers a concrete expression of ideas that can be used as a focus of discussions. For this reason we looked at how designers in other disciplines work with ideas.

Most designers start by hand-sketching ideas. Low-fidelity tools (pen and paper) are preferred because there are no constraints or off-task decisions required (Gross 1998). However typical computer-based design environments require widget selection, placement, sizing and alignment. This has been shown to interfere with the design process (Goel 1995) because the designer is distracted from the meta task by the requirement to make decisions before they are appropriate such as: the choice between a radio button or a check box, and unimportant detail such as alignment. Designers have also found that the tidy product of computer tools implies a higher level of commitment to the design, for both the designer and clients, than is intended and that the finished appearance of computer produced designs means that fewer

changes are entertained (Wong 1992). Sketching allows designers to work quickly with ideas and while doing so order and structure the problem and solution space (Tversky 1999). Working with a sketch also prompts thought about the underlying functional requirements.

There are striking similarities between the way expert programmers describe program creation and how designers work. Lammer (1996), interviewed a number of well known programmers; they consistently describe programming as an art, skill and science, for example Charles Simonyi said "The first step in programming is imagining" (Lammers p. 15) . Lammer's book also includes a number of sketches that are the original designs for software the interviewees had written; from this we can conclude that some expert programmers express their designs as sketches.

Clearly the first task when writing a program is to understand the problem. Novice programmers often have difficulty with this fundamental step, even for simple tasks. Creating a program requires the deconstruction of a problem into its composite parts and reconstruction of it as an algorithm; this abstraction from reality is difficult. One way to get a better understanding of a problem is to work with scenarios. Scenarios provide concrete examples of an abstract problem. Carroll (2000), has written extensively on this technique for defining complex problems, and Rettig (1994) describes scenario based techniques he has used successfully with students to improve their understanding.

Learning to program is a learning task in much the same way as any other learning. Learning with peers in small groups is often more productive than being 'taught'. In fact Vygostsky (1978), contends that most learning is from peers rather than teachers. Also learning is most effective if there is quick evaluation and reinforcement. This is referred to as the experiential learning cycle which has been describe by a variety of authors (e.g. Kolb 1984) and can be summarised as "Do, Review, Revise and Reflect". These general ideas are at the heart of constructivist theories of learning that contend that each of us must construct our own knowledge and that theory together with practice are the most powerful learning experiences.

A number of others have created software to support hand sketching of designs. Landay and others (Lin, Newman et al. 2000; Landay and Myers 2001) created Silk for interface form design and Denim for web page design. Silk is a standalone form design tool that recognises a range of widgets and includes a storyboard and run mode. In run mode some of the widgets have functional behaviours, for example scroll bars can be moved up and down. Denim is a web page design tool; it includes five levels of zooming to aid with the design of page hierarchies and navigation. Both of these tools support conversion into other formats. Knight is a UML diagramming tool developed by Damm, Hansen et al. (2000) that is integrated with the WithClass CASE tool, it includes an interesting mix of formal, semiformal and informal representation in the one diagram. CASE diagrams can become very large so this tool includes a radar window to aid navigation around the design space. Bailey, Konstan et al (2003) have created a sketch tool for multimedia applications' design. This tool supports the inclusion of other media such as pictures or sound bites. They have also put more emphasis on supporting functionality in run mode with navigation and the ability to play the media.

3 Software Development

The goal is to provide an environment that retains the informality of the low-fidelity approach while providing the support expected of computer environments, and for the sketching software to be an integrated part of the programming IDE.

A digital whiteboard provides a space where a small group can work together sharing the image and interaction, and is likely to be able to provide the editing functionality and document management that is expected for computer applications such as cut, copy, paste, resize, and save.

Ideally student programmers should be able to move freely along the design continuum from informal, high-level design, to detailed formal design. The informal environment should be pen-based and put little constraint on what can be drawn.

One of the main complaints students have about pre-sketching their interfaces is that it is 'a waste of time'. By integrating the sketching environment into the IDE and the software intelligently interpreting the sketch so that it can be converted into the IDE form designer it is likely that students will see it as a valuable way to work with no time overhead.

Using scenarios as concrete examples of problems helps students to clarify the problem constraints. We suggest that a computer based sketch environment should allow the user to easily check a sketch with scenarios.

Taking these points into consideration, we have developed an integrated sketch interface (Freeform) for Visual Basic™ (VB). Sections 3.1 and 3.2 describe the development of our prototype system

through two major iterations and the usability study carried out between the iterations. In section 4 we describe a comparative study we conducted of students designing user interfaces using Freeform or a normal whiteboard.

3.1 First Prototype

With the first prototype of Freeform we were mostly concerned with the technical feasibility of the project and in providing a platform to undertake a preliminary usability study. In this section we describe the physical interface, the software and summarise the findings of the usability study of this prototype.

To support this work we have constructed a low-cost large interactive display screen (LIDS) (Apperley, Dahlberg et al. 2001). It is comprised of a standard data projector, rear projected onto a screen approximately 900mm wide by 1200mm high with a Mimio whiteboard digitiser attached to the screen to provide the interaction.

The Mimio pens are used in mouse emulation mode where the pen nib provides left mouse down, mouse move and mouse up events to the program. Although a right-mouse button and double clicks are supported by the Mimio interface, they are cumbersome to use. We built the program so that all interaction is via left-mouse pen actions.

In this environment there is no passive pen tracking as there is with a mouse. It required some experimentation to structure simple interaction, particularly for editing. Early testing indicated that a conservative multi-step approach was easier to learn and use. While the general setup works well, some users commented that the pen was too large and there was too much play in the nib switch which made it difficult to write with.

The software has three major components; a sketch space, recognition engine, and a VB form creator. The sketch space is a deliberately minimalist environment where the users can draw, write and edit. In drawing mode the user can sketch freely but should ultimately end up with glyphs that roughly depict the VB controls that they wish to represent (Figure 1). In handwriting mode the user pens text that is interpreted as a label or caption.

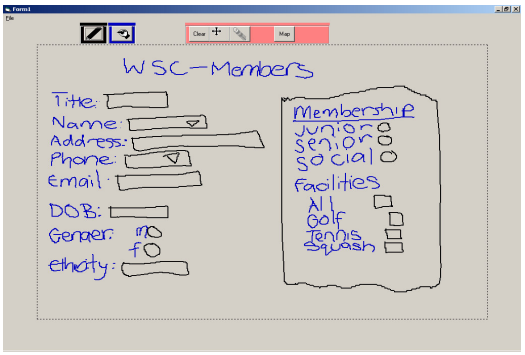

Figure 1 Sample student sketch using first prototype

The recognition algorithm we implemented (see below) requires shapes to be drawn in a single stroke; we found that users had no difficulty with this. We separated drawing from writing to help with recognition; this did cause some initial confusion for users and ultimately we would like to eliminate this modality.

Others have used gestures for editing functions, such as undo, copy or delete; we implemented only a delete gesture. We had overloaded the gesture (Figure 2a) using it for delete and as a text holder; this caused confusion for both the users and the software. For the second prototype we used a different gesture for delete (Figure 2b).

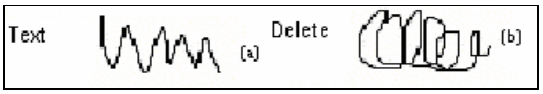

Figure 2 Text and Delete Gestures

In edit mode the user can move, cut, copy, paste or resize sketch components. In this mode each sketch element (individual strokes for drawing ink or words for writing ink) is surrounded by a bounding box. Users can select a single element by clicking inside its bounding box (Figure 3a) or a group of elements by lassoing them (Figure 3b).

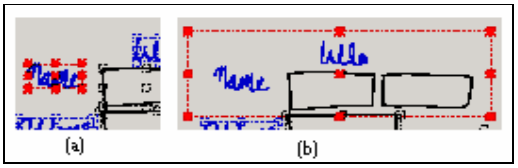

Figure 3 Single and multiple element selections

The bounding box of a selection is highlighted and has perimeter handles and a central handle. The centre handle moves the selection, while perimeter handles resize. Although these functions work well, one of the most requested enhancements was an undo feature. We also decided that being able to convert

ink from drawing to writing or visa versa would be useful until we can integrate the two inking modes. Our goal is to provide an intuitive design environment where all the interaction is via the pen and LIDS; the first usability study indicated that we were well on the way to doing this.

Pen strokes are recognised immediately after completion. Most sketch systems provide immediate feedback by, tidying the sketch, changing the colour or displaying the name of the recognised glyph. Except for the delete gesture, which invokes an immediate response, we chose to delay disclosure until the user decides the sketch is complete so as to not interrupt the design process. Some users need to experiment to feel confident that recognition is going to happen, but having done this they are comfortable with leaving the recognition until later.

The recognition engine has two parts; gesture recognition to classify the stroke, and a rule base to combine strokes and map them to VB controls. Rubine's (1991) algorithm is used for the stroke recognition. A library of classes of shapes for drawing and writing are maintained by the software for matching against; these are fully exposed to the user who can add, change or delete shapes and classes of shapes. Students were happy with the performance of the shape recognition; over the usability study we achieved a 90% success rate. We found the recognition rate for letters was very poor so writing is left unrecognised. Achieving satisfactory word recognition became a goal for the second prototype.

When the design is finished and the user wishes to create a VB form the rule base is used to establish the relationships between ink strokes. There are three categories of relationship; combined strokes, containers and single stroke gestures (Figure 4). Two strokes can be combined to create one VB control; for example a long rectangle with a triangle inside will be mapped to a drop-down list. Container controls such as frames are particularly important as VB uses containers to create mutually exclusive sets of option buttons.

A user interface is provided to the rule base which allows the specification of any intrinsic VB control (Figure 5)**.** In the left section of the form the user specifies the control type and whether this is to be a single stroke, joined or container control; multiple specifications of a control are possible. In the middle section the user specifies the primary shape for the control, the relationship with subsidiary shapes (such as beside, below) and whether they are required or optional. The right section specifies how the VB control properties will be generated. For example the top position of the control can be set as the top position of the primary or secondary shape or the topmost point of either shape.

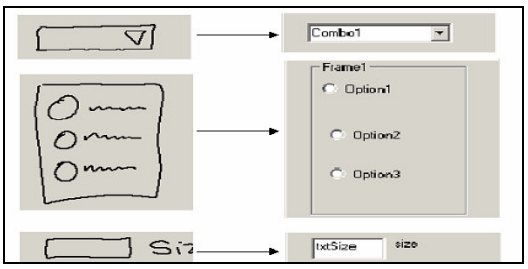

Figure 4 Sketch glyphs to VB controls

We did not usability test this part of the system and have not endeavoured to include checking for ambiguous definitions or other user errors.

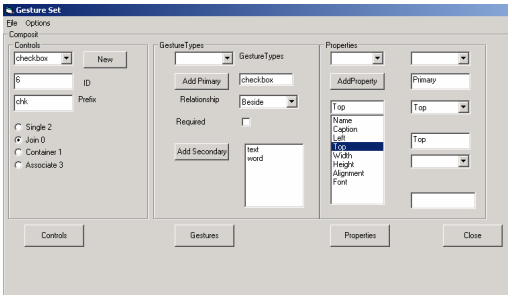

Figure 5 Sketch to control mappings

Before the form is created the user has the ability to alter the recognition engine's decisions by choosing the type of control from a list. Once checking is completed a VB form is generated (Figure 6). While the VB form accurately represents what the user has drawn it looks untidy. A set of controls that look the same on the sketch result in a set of different looking controls on the VB form. This is contrary to our expectations in a formal environment. Standardising the sizes and aligning controls on the VB form became another goal of the second prototype

The overall comments from the students were positive, but they also provided a number of issues to be addressed

- Hardware – a better pen (outside the scope of this project)
- Drawing/writing – changing modes is distracting. Either eliminate modes or make it possible to change the ink mode
- Editing – provide undo
- Recognition – recognise writing
- Transformation – tidy the VB form by standardising sizes and aligning controls.

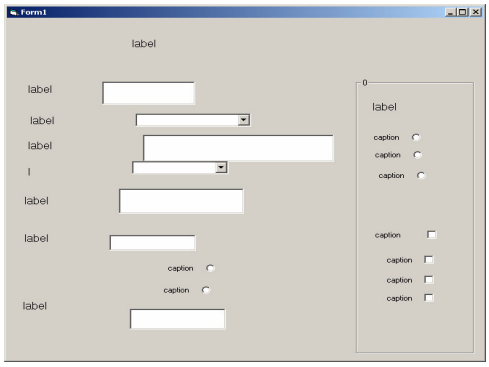

Figure 6 VB Form created from Figure 1 sketch

3.2 Second Prototype

The second prototype expanded the system to provide for multiple sketches and a storyboard view. A run mode was also added to allow users to interactively check their sketches. In addition we addressed most of the issues that were identified in the usability study described above. We acknowledge that the Mimio pens are not perfect for this type of interface; improvements at this level are being addressed elsewhere. This section describes the changes to the software and the repeat usability study.

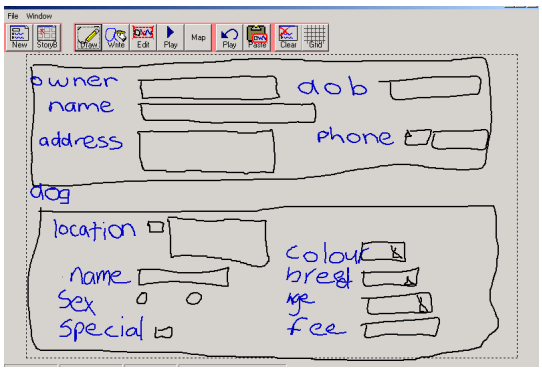

Figure 7 Sample from second prototype

The sketch space (Figure 7) is based on the first prototype but includes added functionality for editing. An unlimited undo function was added; each sketch has its own undo stack. We also added a clipboard so that sketch elements can be copied and pasted on to the same or a different sketch. We were unable to integrate writing and drawing modes (see below) so have provided the ability to select ink and change it from writing to drawing or visa versa. A grid was added to aid form transformation (see below); this can be displayed in drawing mode. The undo and the ability to change the mode of ink were well received by users and some commented that the grid made it easier to draw and write.

The storyboard view (Figure 8) is similar to that provided by Silk (Landay and Myers 2001); it provides an overview of all sketches and allows the user to add navigation links between sketches that can be used in run mode. Because the storyboard has quite limited features (adding, moving or deleting links and moving or deleting sketches) we were able to provide a modeless interface. On a pen down action the software determines whether the pen is positioned on a navigation link endpoint and if it is assumes a link move. Either end of a link can be dragged to a new position, or the trashcan to delete the link. A drag from any other point in a sketch will either create a new link if the terminating point is in another sketch or move the sketch if the termination point is in an empty slot. A sketch can also be trashed by dragging it to the trashcan. This interface proved to be very easy and intuitive to use.

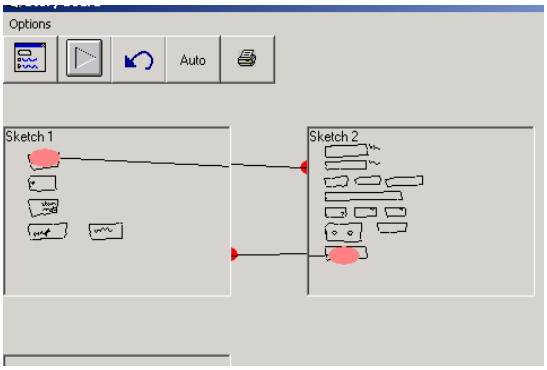

Figure 8 Storyboard View

The run mode (Figure 9) facilitates active checking of designs; in this mode it is as if a transparent overlay is placed over the sketch with hotspots at the source points of navigation links. The underlying sketch is inert. The user can draw or write on the overlay and navigate between forms by clicking the hotspots. The run mode ink can be cleared, but not edited. It stays on the sketches while the user is in run mode, but is erased when the user returns to the sketch space. While this did not cause any comment during the usability study in the larger evaluation that followed it was clear that it would have been better not to remove the ink, but simply to hide it when the user returned to sketch mode.

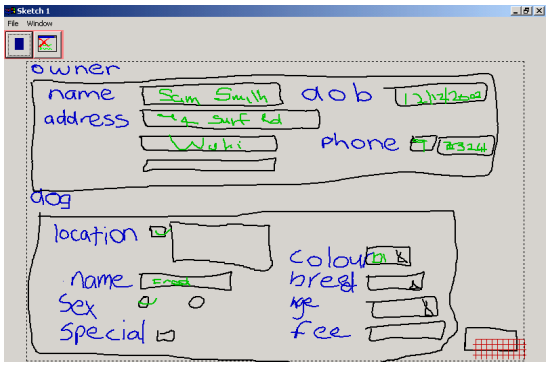

Figure 9 Sketch in Run Mode

We achieved limited word recognition by matching words against a vocabulary, adding two extra features to Rubine's Algorithm (1991), and limiting input to lower case letters. We compiled a vocabulary by extracting captions from hundreds of sample VB programs. We observed that the algorithm often confused letters like 'b' and 'd' or 'm' and 'w' so we added features which gave the x and y point-of-balance of the ink stroke. The algorithm produces a list of probable letters for a given stroke in decreasing order of probability. We then match the input word against the vocabulary. The word from the vocabulary with the lowest mean letter position from the probable letter lists for each stroke is the successful match. If the best match has a mean letter position greater than three then no match is found. The recognition rate for words is not high; we can achieve about 70% recognition of words that are in the dictionary, using our own training set and being careful to form letters correctly. Students during the studies achieved much lower recognition rates. However it is easy to correct words by selecting from the vocabulary and while we would like to achieve a better result, users were prepared to work with this.

We also improved the tidiness of the VB form (Figure 10) by aligning controls to a grid and adding to the rule base to standardise the sizes of controls. The grid size can be changed by the user; we found 400 to 600 twips (30 - 40 pixels) worked best. The software aligns the top left corner of each ink stroke's bounding box to the closest grid intersection point. The rule base allowed for any control property to have a fixed value or a unit value. For example radio buttons can be set at a fixed height of x pixels and edit boxes can be set to have a height that is a multiple of y pixels. When calculating control sizes we rounded all values down as experience indicated that most sketch elements were larger than those required on a form. The forms from this prototype were much more satisfactory and users were generally pleased with the results.

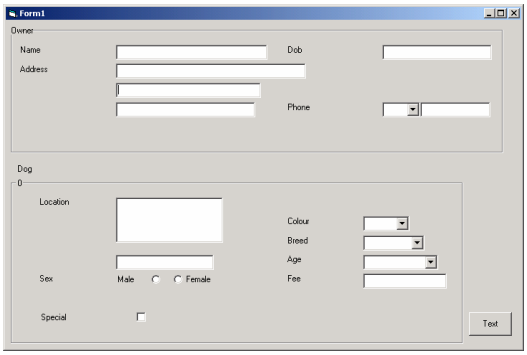

Figure 10 VB form from Figure 6 & 8 sketch

4 Evaluation Study

We used the second prototype to evaluate the use of such an electronic sketch tool as a design environment for student programmers. We conducted a study where eight small groups of students (2 or 3) completed two interface design tasks. All of the students were from a first-year VB programming course.

For one task they used the Freeform environment described above and for the other they sketched a design on an ordinary whiteboard and then created a VB form from their design in the normal manner. The problems were designed to be of a similar type and difficultly, one was a simplified book catalogue form, the other a dog registration form.

We evaluated the study through; participant questionnaires, review of the design products by an independent expert, observation, and review of the learning process by an educational psychologist.

We questioned the students about their experience before the study and after each task was completed. Before the study they answered three questions about: their level of experience of using whiteboards for design task, their belief on the usefulness of sketching designs, and their current practice of sketching designs. While most thought that sketching designs was a good idea, there was no correlation between this and their current practice, with few hand sketching a design before they created the interface in VB.

After each task was completed the students answered ten questions on their enjoyment, problem understanding and the ease of use of the environment. Statistical analysis of the co-variance between the mean group responses for the two tasks

indicated at a significant level of greater than 95% that:

- they enjoyed the Freeform task more
- it increased their motivation to learn programming
- they would like to use Freeform as a program planning tool in the future.

A further two were more positive for Freeform at a greater than 90% level:

- they felt prepared to complete the program
- they found checking the scenarios was easy

Of the remaining four questions two were most influenced by the order of the task, with the second task being easier, and the other two questions were higher for Freeform but not at a statistically significant level. Seventeen of the twenty participants stated that given a choice of a standard whiteboard, Freeform, or nothing, Freeform would be their preferred design environment. Finally a comparison between their view of the value of sketching before the study and after each task showed that Freeform gave a significant boost to their rating of the importance of sketching designs.

We observed that students made many more changes to their sketches in the Freeform environment. Most of these changes were made after they had checked their designs in run mode. The changes resulted in the designs created with Freeform being more appropriate solutions for the problem (our independent expert scored most groups' Freeform design higher). A typical example of a change that was made after checking in run mode was the space for address data required by one of the problems; most groups initially drew a single-line edit box. Three out of the four groups who did this problem using Freeform finished with space for multiple address lines. Only one of the four groups who did this problem on the standard whiteboard finished with space for multiple address lines.

We asked an educational psychologist, to review the video tapes; he thought there were a number of possible reasons for the increased changes and therefore better designs created in Freeform. He suggested that there is a lower cost and lower risk in the electronic environment. Changes are lower cost because of the ability to move and resize existing elements and lower risk because of the undo facility. He also suggested that the run mode encouraged active participation in the checking process where the normal whiteboard checking was more passive. Along with this the immediacy of the sketch and check with the electronic environment provided quick feedback and completion of the learning cycle which is likely to encourage more activity.

Another typical change that was made more frequently in the electronic environment was changing an edit box to a drop down list. Students told us that the electronic environment made them think more about the functionality of the program than the whiteboard environment and this is why they made more of these types of changes.

5 Discussion and Conclusions

This project has brought together ideas from HCI, design, usability engineering and educational psychology to develop a new computer supported environment for informal interface design.

Research from other disciplines suggests that hand drawing initial designs is preferable to formal computer design tools that require selection and placement of widgets because informal diagrams allow the designer to leave parts of the design ambiguous and the unfinished look of a sketch encourages change. From this comes our commitment to a simple sketch space where the user is not distracted by notification of the recognition engine interpretation of ink.

Our software is intended for small group use on a public-space (whiteboard). We have noticed some differences in our approach to inking and recognition in comparison with private-space pen interfaces such as PDAs. We have put an emphasis on maintaining the image and in-place inking that is not required in a private space. This precluded us from using such techniques as the letter-by-letter type recognition that is common on PDAs. Better handwriting recognition is an outstanding challenge.

From usability engineering and scenario based design we have adopted the ideas of being able to check designs while they are still in sketch form. The run mode we added to the second prototype was very effective at engaging the students in the checking process and resulted in better designs.

We choose to evaluate our environment against a static sketch environment and have found some significant advantages in the computer supported environment, particularly for checking sketches. How our environment would compare with checking sketches in a normal IDE form designer is something that should be investigated.

From an educational perspective we have provided a shared work space where small groups of students can work cooperatively on a problem. They enjoyed working with a novel tool and the ease of checking and changing the sketch along with the

rapid feedback encouraged the students to alter the form as they thought more carefully about the problem requirements and better ways to provide an easy-to-use interface.

We made decisions about how and where to integrate sketching into the programming IDE. It would be possible to add sketching to the main form design space of an IDE. We chose to remove it so that there was a definite feel of being somewhere else and working in a different mode. Sketching software could also be a standalone tool; however we liked the idea of students being able to move along the design continuum with a minimum of disruption. The current version of Freeform does not change a sketch to reflect changes in the IDE design environment; this is an enhancement we would see as being useful.

This software has been developed and evaluated for VB, but the principles should hold true for other similar programming IDEs such as Delphi, C++. In fact, as the recognition libraries and rule-base are exposed to the users it would be a simple task to adapt the software for use with other types of tools. Our tool supports only the intrinsic VB6 controls; it would be possible to extend to more complex controls, however the rules required to differentiate widgets become more complex as the number of widgets increase.

In conclusion we believe that public-space, informal design environments have real potential for classroom use. Students enjoy and learn by working together and the ability to provide editing facilities and other modes such as the run mode we implemented in the second prototype can add positively to the learning experience.

Acknowledgements

The assistance of Matt Jones, Ray Littler and Tony Morrison for their help with respectively: the evaluation of the designs, statistical analysis, and comments on the learning processes, and the many students who have contributed to these studies is gratefully acknowledged.

6 References

Apperley, M., B. Dahlberg, et al. (2001). Lightweight capture of presentations for review. IHM-HCI, Lille, ACM.

Bailey, B. P. and J. A. Konstan (2003). Are Informal Tools Better? Comparing DEMAIS, Pencil and Paper, and Authorware for Early Multimedia Design. CHI 2003, Ft Lauradale, ACM.

Carroll, J. M. (2000). Making Use: Scenarios and Scenario-Based Design. Symposium on Designing Interactive Systems.

Damm, C. H., K. M. Hansen, et al. (2000). Tools support for cooperative object-oriented design: Gesture based modelling on and electronic whiteboard. Chi 2000, ACM.

Goel, V. (1995). Sketches of thought. Cambridge, Massachusetts, The MIT Press.

Gross, M. (1998). The proverbial back of an envelope. IEEE Intelligent Systems**:** 10-13.

Kolb, D. A. (1984). Experiential Learning: Experience as the Source of Learning and Development. New Jersey, Prentice-Hall Inc.

Lammers, S. (1996). Programmers at work: Interviews with 19 programmers who shaped the computer industry. Redmond, Microsoft Press.

Landay, J. and B. Myers (2001). "Sketching Interfaces: Toward more human interface design." Computer **34**(3): 56-64.

Lin, J., M. W. Newman, et al. (2000). Denim: Finding a tighter fit between tools and practice for web design. Chi 2000, ACM.

Rettig, M. (1994). "Prototyping for tiny fingers." Communications of the ACM **37**(4): 21-27.

Rubine, D. (1991). Specifying gestures by example. Proceedings of Siggraph '91, ACM.

Tversky, B. (1999). What does drawing reveal about thinking. Visual and spatial reasoning in design, Cambridge USA.

Vygostsky, L. S. (1978). Mind in society: The development of higher psychological processes. Cambridge MA, Harvard University Press.

Wong, Y. Y. (1992). Rough and ready prototypes: Lessons from graphic design. Human Factors in Computing Systems CHI '92, Monterey.