
THE UNIVERSITY OF
WAIKATO
*Te Whare Wānanga o Waikato*

# Using Shared Displays to Support Group Design; A Study of the Use of Informal User Interface Designs when Learning to Program

Beryl Elizabeth Plimmer

This thesis is submitted in partial fulfilment of the requirements for the degree of Doctor of Philosophy in Computer Science at The University of Waikato.

May 2004

## Abstract

Hand-drawn sketches have traditionally been used to depict design ideas because they are quick to draw and can include as much or little detail as is required to convey the essence of the ideas. Computer tools are now an alternative and offer advantages for editing, storing and transmitting designs. However, designers consistently reject using current computer tools because these tools interrupt the creative process. Various studies have supported the designer's position, consistently showing that traditional tools produce more and better design ideas.

This thesis describes the development and evaluation of a design-friendly computer tool that focuses specifically on the needs of the novice programmer who is designing user interfaces. From an extensive review of the literature on design, learning to programming and previous sketch tools we extracted the specifications for a tool that: compares favourably as a design medium with traditional tools such as the pen, paper and whiteboards, provides the editing and storage support expected of computer tools, helps students to gain a better understanding of programming problems and integrates seamlessly into a program development environment.

Freeform, the tool we have developed to these specifications, has had two iterations of development and usability testing. This tool is unique in that it: is integrated into a commercial program development environment, uses a digital whiteboard for interaction and includes character recognition. Using Freeform, students can both quickly hand-draw user interface designs and interact with the design while it is still rendered as a sketch. When satisfied with the design the student invokes the recognition engine. The sketch is then overlaid with recognition data. Any incorrect interpretations can be altered by the student. The student can then instruct Freeform to create the formal user interface in the program development environment. The translation of sketch glyphs to user interface widgets is achieved by parsing the sketch and recognition data with transformation rules.

We have conducted two evaluation studies using Freeform. The first study directly compared Freeform to a traditional alternative. We found that, although the design quality was similar, the students believed that when they were using Freeform they both understood the problem better and enjoyed the experience more. We noted during this study that the interactive checking available in Freeform prompted more changes to the designs than the static checking done on a standard whiteboard. In the second study, we asked students to check designs rendered as both sketches in Freeform and as formal diagrams in a user interface builder. The students made significantly more revisions to the Freeform sketches and therefore produced better designs from Freeform.

The usability tests and evaluation studies we have conducted suggest that computer-based low-fidelity design tools: can compete favourably with traditional tools as a design medium, offer better support for editing and storage, and may have advantages for checking over both traditional sketch mediums and formal interactive computer designs.

## Acknowledgements

To the many people who have supported me during this journey.

Grant, and our children Karen, Rob, Nick and the rest of the family.

Supervisors Mark Apperley, Masood Masoodian and Matt Jones.

Friends and colleagues.

My students who were the motivation for this work, and many of whom participated in the various studies.

You have all helped in so many ways. I could write a very long list but Grant has told me that that is just plain boring! I have bored you all quite enough over the last n++ years, so I will just say again thank you all for your love, support and help.

## Table of Contents

## List of Figures

# List of Tables

# Chapter 1
# Introduction

Hand-drawn sketches have been used to communicate design ideas for centuries and most designers still prefer hand sketching their initial designs. Computer-based design tools are now an alternative and offer advantages for editing, storing and transmitting designs. However, designers are reluctant to use computer tools (Bailey et al., 2001b; Damm et al., 2000a; Do & Gross, 2001; Goel, 1995; Landay, 1996; Newman & Landay, 2000) as current technology adversely effects the design process (Bailey & Konstan, 2003; Black, 1990; Goel, 1995). Creating a design with a computer tool is slower than sketching and the designer's attention is continually diverted from high-level conceptual ideas to decisions involving details such as widget type (Goel, 1995). Also, informal sketches are better for eliciting the high-level critique of a design because the finished appearance of computer-rendered formal diagrams implies completeness and this discourages reconsideration of fundamental decisions (Wong, 1992).

These advantages and disadvantages of low-fidelity (pen and paper) and high-fidelity (computer tools) hold true across a wide range of disciplines, one of which is computer interface design. However, computer interface design is somewhat unique in that it is often undertaken by programmers who have little experience of design theory and who see the advantages of a computer environment without recognising its disadvantages (Black, 1990). Advances in pen input devices provide a platform for computer tools that honour the traditions of informal design.

## 1.01 Motivation

The normal process of creating a design is to work first with informal abstract sketches to quickly express ideas. In disciplines where the design product has functionality, for example engineering, architecture or computer interfaces, the visual appearance and the behavioural requirements are developed in parallel. A process of iterative evaluation and refinement continues in the informal environment until the detailed design is nearly complete. Once all the major decisions have been made the designs are transferred to a computer-based design tool for formalisation. In the case of computer interfaces the computer tool is often the form builder of the integrated development environment (IDE) which formalises the design and, at the same time, creates the design product.

The informal early design process is often undertaken by a small group of designers who work on a whiteboard.

> We sought to explore the utility of replacing low-fidelity media (whiteboard or paper) with a large shared screen that accepts both direct pen input and the software that provides the appropriate support.

In order to do this we have created a computer-based design tool (Freeform) that honours the traditions of low-fidelity environments while providing support for editing and saving documents expected of computer tools.

While other sketch tools have been targeted at commercial designers (for example Bailey & Konstan, 2003; Landay, 1996), Freeform focuses on the needs of novice programming students. In many ways the needs of novices are the same as those of experienced programmers, yet additionally, they face both the challenge of learning to program and that of creating a usable interface. User centred and scenario based design methodologies promote the use of low-fidelity interface design as a central part of understanding problem requirements and as such are recommended techniques for novice programmers (Carroll, 2000; Rettig, 1994). This type of approach to problem solving where concrete examples are examined before the abstract representation is composed is also consistent with constructivist learning theories (Kolb, 1984). These ideas are central to Freeform

as they encourage exploring the user interface requirements as a first step of program development. One of the objections of students to low-fidelity prototyping is that the sketch is thrown away. This led us to integrate the Freeform tool directly into a programming IDE with the sketch automatically converted to a formal design in the IDE form designer.

The requirements of a computer-based sketch tool are:

- to retain the informal, direct, unconstrained nature of pen and paper/whiteboard for the creation of hand-drawn diagrams
- to provide a drawing space that is large enough for a group to share
- to provide support for editing and storage of sketches
- to include a way for users to emulate the use of their design sketches
- to integrate the software with the formal design environments used at the next stage of software development

Others have evaluated low-fidelity sketches against high-fidelity computer drawing tools and consistently found that sketches are superior in terms of speed, creativity and the communication of ideas (Bailey & Konstan, 2003; Black, 1990; Goel, 1995). Bailey and Konstan (2003) undertook a three way evaluation of: hand sketching using pen and paper, a formal computer tool (Authorware) and their computer-based sketch tool (Demais) with experienced designers. In general their study participants still preferred the low-fidelity medium but the computer-based sketch environment compared well in regard to communicating functional requirements. We evaluated Freeform as a design tool against a whiteboard and as a medium for checking designs against an IDE. These evaluations indicate that such environments are likely to be useful to novice programmers and suggest that sketch-based computer tools are likely to have less effect on the design process than their formal predecessors.

We see the potential benefits of a computer-based sketching tool as

- a designer friendly environment

- with support for editing, storing and transmitting artefacts
- support to prototype interactive behaviour while the design is still rendered informally and design ideas are fluid
- seamless integration during the design process from informal to formal rendering of designs

The goal of this work is to explore the use of large shared-space design tools for computer interface design. In order to do this we have developed a tool that concentrates on the requirements of novice programmers and evaluated it using this target group. We contend that although novice programmers have some unique requirements, the challenge that they face with problem understanding continues to be a challenge to experienced programmers, the major difference being the size and complexity of the problems. Therefore, by studying novice programmers we are likely to be able to offer suggestions as to the utility of computer supported sketch environments for design tasks in general and, more specifically, for both user interface design and as a tool for novice programmers.

## 1.02 Organisation of thesis

There are four main sections to this thesis: the first part, chapters 1 – 3, includes this introduction, a review of relevant literature and the conceptual tool design; the second part, chapters 4 – 7, describes the development and usability testing of Freeform; the third part, chapter 8, describes the evaluation of the utility of Freeform as a tool for novice programmers; finally, chapter 9 reviews and discusses the work and suggests future work. More detail of the content of each chapter is given below.

Chapter 2 provides the background to this work. There are three main sections in this chapter; early design, learning to program and computer-aided design tools. Early design looks at the design process in general and in particular at the role of sketching in early design. Traditionally pens, pencils, paper and whiteboards or blackboards have been used for creating prototypes. There are many hi-fidelity computer-based design environments targeted at a wide range of domains (e.g.

CAD, CASE, IDEs). Studies have shown that these environments interfere with the design process. A review of these studies provides pointers to the essential elements of low-fidelity tools that need to be preserved in a computer-based environment.

The second section of the literature review relates the learning of programming; the target users for this study being novice programmers. It examines the skills and knowledge needed to be a successful programmer, and also looks into the teaching and learning of programming. Programming, as a discipline, is changing rapidly. The IDEs that are the predominant program creation environments of today are relatively new as are GUI user interfaces. The newness of these technologies means that we must extrapolate from research on earlier programming environments and styles. This section also considers where user interface design is incorporated into software development lifecycles and how this may inform the software development process.

The final part of the literature review focuses on sketch design tools that others have developed. This section is divided into five sub-sections; the physical interface, sketch space, recognition, emulating behaviour and the transformation from sketch to formal environment.

Chapter 3 draws together ideas from the literature to describe the requirements of a sketch tool that would support novice programmers. It discusses how such an environment may be incorporated into the program design process by student programmers. The look and behaviour of the software are described and there is discussion of some of the alternative approaches that can be taken to critical issues such as recognition, and the tradeoffs and assumptions that need to be made to move from a conceptual system to implementation.

Our software development methodology was a blend of prototyping and rapid application development. We completed two major prototype iterations: the first to prove the technical feasibility of the project and provide a basic interface to elicit usability requirements, the second incorporated additional functionality and a more sophisticated user experience.

The first prototype of Freeform is described in Chapter 4. Freeform is a Visual Basic 6 (VB) plug-in and was written in VB6. VB is a typical IDE and was chosen because it is often used for novice programming courses. The main goals of this prototype were to prove the technical feasibility of the project and to provide a platform for initial usability testing. This prototype consisted of a sketch space, recognition engine and the rule-base to generate a VB form from the recognised sketch. There were some significant technical challenges in integrating the tool into VB6, for the most part because it is not possible to create new, independent controls within a running VB6 program (Marsden, 1997).

Chapter 5 describes the usability test carried out on this prototype and the subsequent enhancements planned for the next prototype. The biggest challenges to come out of this usability test were the user's need for character recognition and a tidier VB form. This prototype was reported during development and after usability testing (Plimmer & Apperley, 2001a, 2001b, 2002a).

The second prototype is described in Chapter 6; it incorporated the findings from the usability study and most of the requirements described in Chapter 3. The usability of the tool and the sketch to VB form transformation were improved, and multiple sketch spaces, a storyboard, character recognition, and interactive checking were added (Plimmer & Apperley, 2002b).

A further usability test was conducted on this prototype; it is described in Chapter 7. This usability study suggested that Freeform was sufficiently developed to allow us to ascertain its utility as a tool for novice programmers (Plimmer & Apperley, 2003b).

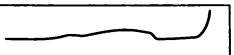
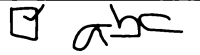
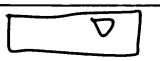
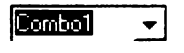
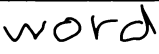
The usefulness of Freeform to student programmers was determined by means of the evaluation studies described in Chapter 8. Two studies have been undertaken. In the first, small groups of students were asked to design user interfaces for two problems. For one task they used a standard whiteboard and the VB form builder; for the other they used Freeform and automatically converted their sketch into a VB form. The analysis included student questionnaires, an expert review of the designs, an expert review of the learning experience, and observations of the writer. This study was reported at CHI 2003 (Plimmer & Apperley, 2003a). One

significant difference found between the environments was that there were a higher number of changes made after checking the designs in Freeform. A second study was undertaken to find out whether this was because of the interactive nature of the checking in Freeform or because of the informal appearance of the design (Plimmer & Apperley, 2004). For this study small groups checked two designs; one was presented as a formal design in the VB IDE and the other was presented as a sketch than Freeform. Students made significantly more changes to the sketch in the formal design. This is consistent with Wong's (1992) suggestion that the finished look of formal diagrams limits the changes people make.

Finally, Chapter 9 reviews the work and discusses what we have learnt about the utility of large shared-space design tools in general and specifically how they maybe useful to novice programmers. It also suggests further research that could be undertaken (Plimmer & Apperley, 2003c).

## 1.03 Definition of Terms

A number of different words are used to describe drawing and writing actions, and artefacts in everyday language. For clarity, the specific terms used in this thesis and the interpretations ascribed are listed below.

| | | |
|---|---|---|
| Stroke | a user pen-down, pen-move, pen-up sequence |  |
| Mark | a general term for a completed stroke, includes both gesture and glyph primitive |  |
| Gesture | a stroke that invokes an action (this is the delete gesture used in the second prototype) |  |
| Ink | a general term for any hand-drawn symbols/ letters/words that remain on the sketch |  |
| Glyph | a hand-drawn symbol that represents a form control, consisting of one or more strokes |  |
| Widget | a computer generated symbol that represents a form control |   |
| Letter | a stroke or strokes that represent a alphabetic character or number |  |
| Word | a set of adjacent letters |   |
| Icon | screen symbol usually in a button that invokes an action |   |

# Chapter 2
# Design, Learning to Program and Computer Supported Design Environments

User interface design and construction has been an important aspect of computer programming since the advent of visual displays and direct keyboard input. End-user computing, personal computers, the internet, mobile computing and ubiquitous computing have placed even greater emphasis on the user interface. In current systems a large portion of programming effort goes into supporting the user interaction (Myers & Rosson, 1992). By borrowing techniques from more mature design disciplines and correctly applying the design process to software development student programmers will be more successful in building programs that provide a good user interface.

This chapter, to provide the background for the requirements and constraints on a computer-based sketch environment for interface design, reviews literature from three different fields: design, learning to program and previous computer-based sketch tools. Section 2.01 describes the design process in general and in particular looks at the role of sketching as an important aspect of this process. It also examines how collaborative design groups interact with design visualisations such as hand-drawn sketches and computer produced diagrams. This is followed by a description of traditional design tools and surfaces, and computer-aided drawing tools. This section concludes with a summary of the advantages and disadvantages of both traditional and computer-aided environments, as well as looking at the effects of each environment upon the design process.

Section 2.02, which deals with learning to program, reviews (1) the literature on how expert programmers go about creating a program and then (2) existing research on how novices learn to program. Most of this literature relates to

procedural programming although common threads can be observed between creative design and program creation, which are likely to hold true for programming languages with visual interfaces. Following this, both the theoretical models of the software development life cycle and how user interface design fits into these models are reviewed. The next part of this section considers how designing the user interface can inform the requirements analysis phase of program development. Finally, we explore how user interface design is likely to help beginner programmers.

Section 2.03 reports on previous work done on computer sketch tools. It looks at both the physical hardware and the software functionality that has been provided in the sketch space. Intelligent recognition is an integral part of most sketch systems and this, along with work on emulating the behaviour of design products, is described. The final part of this section discusses techniques that are used for the automatic transformation of hand-drawn sketches into the formal diagrams used by computer-aided drawing programs.

# 2.01 Early Design

The process of creative design can be understood as being remarkably similar across a wide range of disciplines, from engineering and architecture to graphic art and user interfaces. Section 2.01.1 describes the design process; considering more closely both the discovery and exploration stages, and collaborative design. Section 2.01.2 describes the tools traditionally used by designers. Following this, Section 2.01.3 examines of how current hi-fidelity computer design construction environments impact upon these activities. Section 2.01.4 compares traditional tools to computer-based tools.

## 2.01.1 The Design Process

Designing in general terms involves planning to make something new and unique. Newman and Landay (2000) describe design as an iterative process broken into four phases: discovery, exploration, refinement, and production (Figure 1). The discovery phase scopes the project defining the requirements and constraints.

During design exploration possible solutions are identified and explored. Some of these ideas for solutions are then selected for the next phase, design refinement, where detail is built up until the design is fully described. Completed designs are then used to create the product. This model is very similar to the traditional waterfall model of the software development life cycle. As with the waterfall model, the more thoroughly the early stages are completed the higher the probability of a successful outcome.

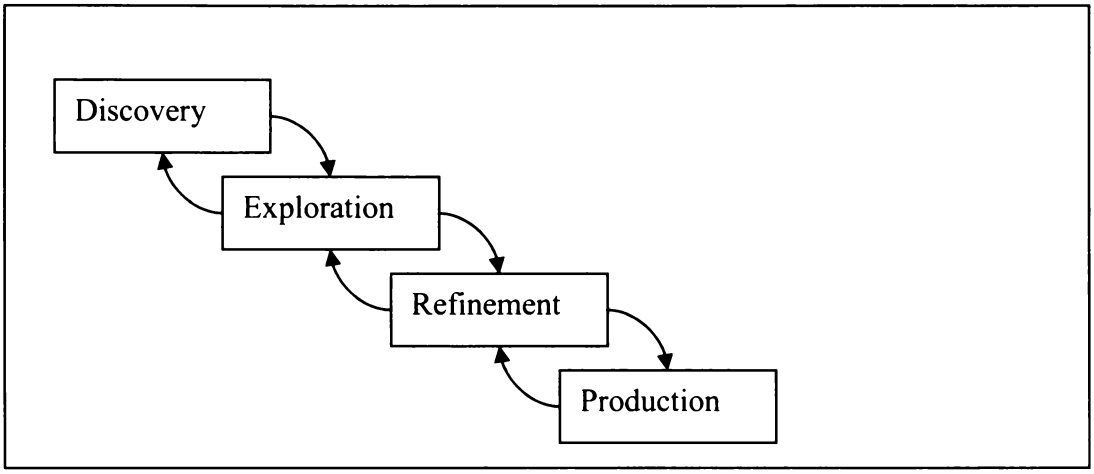



**Figure 1: Phases of Design**

While a design task is initially described in terms of a problem, a goal and constraints, it is likely that none of these aspects of design are fully defined (Goel, 1995). The purpose of the discovery phase is to clarify the design brief. For example, while a client may specify that a form should record phone numbers without specifying details, the designer may need to know whether these phone numbers are to be used for auto-dialling, whether they are fixed numbers, such as home, work and fax numbers or whether there should be the ability to record many different contacts. It is useful to remember that there is not one correct solution to a design task, but rather a large set of possible solutions that may each satisfy the same goal.

Given the design brief (that may not be complete), a designer explores ideas mentally, which he/she expresses externally. *Imagery* is the internal mental representation of a picture or drawing (Goldschmidt, 1991). Designers first conceive of a design as imagery which they may manipulate internally. This said,

the human short term memory is quite limited, making it difficult to mentally create a complete solution to any non-trivial design. To work further the design itself needs to be expressed externally. Sketching or diagramming is the preferred first external representation method (Goldschmidt, 1991; Tversky, 1999).

Gross and Do (1996) draw a distinction between different types of hand-drawings. They define *diagrams* as simple constructions composed of primitive elements chosen from a small set of symbols such as boxes, circles and lines. In contrast, *artists' sketches* use a far wider set of symbols and may have a complex overlay of lines to give a feeling of depth and perspective. A *diagram* can be an untidy hand-drawn sketch or a precise formal plan. For the purposes of this discussion we use the terms *sketch* and *diagram* interchangeably, contending that the early designs of computer interfaces are hand-sketched diagrams.

Tversky (1999) states "design without drawing seems inconceivable". Externalising imagery by drawing it is an integral part of the design process. During the exploration phase a designer may rapidly conceive an idea, express it in a sketch, examine, refine, interpret and perhaps reinterpret it. There is a complex interplay of activities during this phase; new ideas are conceived and existing ideas refined. The sketch may trigger thoughts about the shapes and relationships between design elements, and the underlying functionality required in the working product (Goel, 1995; Goldschmidt, 1992; Tversky, 1999). For example, after first sketching fixed controls for each type of phone number (Figure 2), the designer may decide that being able to select the type of contact and then enter the phone number may be more useful (Figure 3).

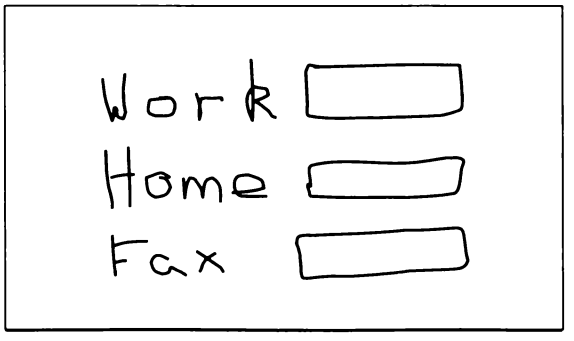



**Figure 2: Phone Number Design One**

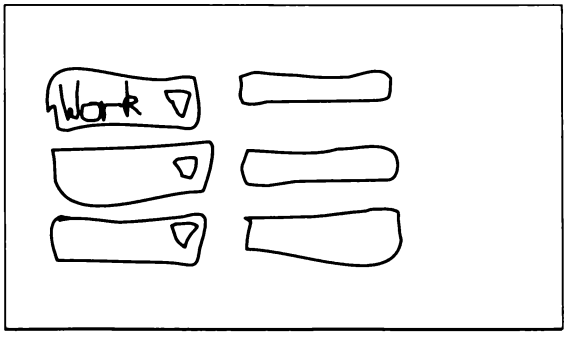



**Figure 3: Phone Number Design Two**

The drawings that are produced during in early design work are imprecise, vague and ambiguous (Gross & Do, 1996). These qualities are important as they stimulate thought about the solution space. The imprecision and vagueness allows the designer to consider different interpretations that may lead to entirely new ideas (Goel, 1995; Goldschmidt, 1992; Tversky, 1999). Decisions about detail can be delayed by using ambiguous glyphs while higher level concepts are resolved. For example, the phone numbers in the figures above may have first been depicted as a single box that acted as a space holder (Figure 4).

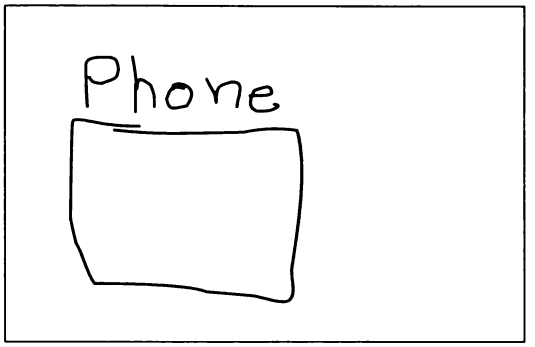



**Figure 4: Phone Number Placeholder**

Any non-trivial problem has component parts. When working with a problem, the drawing process naturally segments the design into recognisable parts, making it possible to see how these components can be recomposed into different segments (Tversky, 1999). Sequence is also implicit in many diagrams. For example, Figure 2 implies that the 'work' phone number is the first point of contact.

Tversky (1999) also suggests that a drawing can depict the underlying organisational and conceptual structure. A great deal of both thought about and expression of the underlying structures occurs during the creation of a design sketch. Because a design is too hard to construct mentally, design ideas are constructed hierarchically (Tversky, 1999), with the larger parts being constructed and then incrementally refined, using the sketch as a prop. Architects and engineers also consider functional requirements while working on the visuals of a design project (Gross & Do, 1996; Stahovich, 1998). It is reasonable to assume that this approach holds true for user interface design.

Many design tasks require group agreement, which may involve a client and a designer or a design team consisting of people from different disciplines. Early

involvement by all members of the group is likely to result in a better understanding of the project and a better design outcome. The process of creating the design, marking the drawing surface and the discussion that accompanies this process, is as valuable as the artefact itself (Bly & Minneman, 1990). Bekker (1993), found that when a group worked together on the initial design, all the members of the group had a better understanding of the problem, solution space and constraints. Tversky (1999), describes how a sketch sets up a visual dialogue with which the group can interact. Bødker et al. (2001), in their work on a design collaboratium, have observed that the group process builds design knowledge for all the participants.

Wong (1992) describes how the feedback received from the group depends on the type of diagram shown to them. For instance, an abstract high-level diagram elicits feedback on high-level design elements; a detailed diagram is more likely to result in feedback on the detail (for example, fonts or colours). Early in the design process, it is important to focus on the high-level elements; hence an outline sketch is more appropriate as it focuses people on the meta-aspects of the design. Wong (1992) also describes how feedback can be elicited on a specific aspect of a design by depicting it in detail while leaving other elements less defined.

Wagner (1990), in describing her 'perfect' design environment, believes the design group should work with sketches rather than more formal designs so that the group is focused on the overall look and feel of the design. She describes an environment where the design group can iteratively design and refine the user interface, slowly adding detail and functionality as the design proceeds.

### 2.01.2 Traditional Sketch Tools

It is likely that people have always sketched as part of the design process, perhaps even with a stick in the sand. However, the first substantive records of design sketches date from the late 15$^{th}$ century, the period when paper first became readily available (Goldschmidt, 1991). Pen and paper is still the most widely used medium for sketching. Whiteboards and blackboards are also popular, particularly in classrooms or when a group is working together on a design.

Each of these mediums has advantages and disadvantages. A design drawn in the sand is easy to change but lacks permanency and portability. Whiteboard drawings are also easy to alter and large enough for a group to share, however, the available space is generally quite small and like sand it is a non-permanent, non-portable surface. In contrast, paper provides more permanency and does not have the same space restrictions. While one can choose a piece of paper of appropriate size for the scale of the task, knowing that extra paper is generally readily available, a paper sketch is more difficult to modify.

Traditional tools do not have the editing functionality for activities such as copying and resizing, which we have become so accustomed to in computer applications. Neither do traditional mediums make it easy to emulate the behaviour of the finished product. However, traditional tools do provide the flexibility that allows designers to draw quickly and freely in a manner that current computer software prohibits.

### 2.01.3 Hi-fidelity Computer-aided Design Tools

Current computer-aided design tools work on the principle of selecting a widget from a toolbox and placing it onto the drawing space. The Visual Basic 6 design environment (Figure 5) is a typical example of this technology.

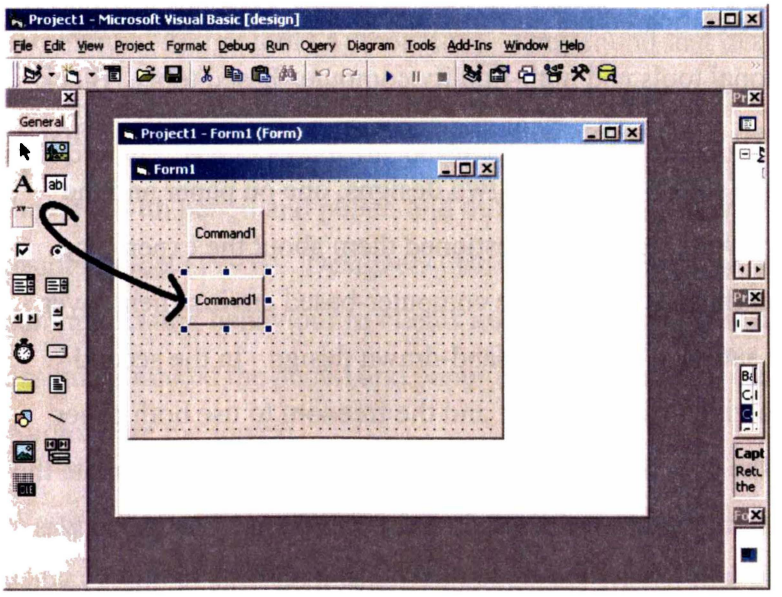



**Figure 5: Example of Computer Drawing Tool**

The designer must select a widget from the toolbox on the left, and then place and size it onto the form. To change the caption of the button showing on the form the designer must then select the appropriate property from a panel on the right and type in the new caption. Fonts, colours and appearance can also be changed via the property panel. The tasks of select, placing and editing design widgets impact upon the design process.

Immediately the designer is forced to focus his or her attention on details which Wagner (1990) and Wong (1992) suggest should be ignored early in the design process. First, the requirement to select a widget from an available pallet requires the designer to decide which widget they wish to use. Often, early in the design process the designer is not ready to make that decision (Gross & Do, 1996). Next, the designer tends to want to match the size and alignment of associated widgets. Also, to alter the text and other properties the designer then needs to shift her or his focus to another part of the screen. All of this takes time and requires cognitive effort. Creating a prototype design with at computer tool typically takes much longer than it does with a hand-drawn sketch (Bailey et al., 2001b; Goel, 1995, Stahovich, 1998 ; Landay, 1996; Rettig, 1994; Wagner, 1990).

### 2.01.4 Comparing Traditional Tools to Hi-fidelity Computer Tools

A number of studies have compared the performance of designers when using both low-fidelity traditional tools and high-fidelity computer-based tools. Black (1990) has conducted two experiments with graphic design students. In the first experiment, she asked the students to create two designs, one on paper and the other using computer software, and then questioned them about the experience. While they were more enthusiastic about the computer-based environment, over time they became more satisfied with their hand-drawn design. In the second study, the students were given a design brief and the freedom to use both paper and a computer. After the task was completed students were asked to estimate the time they had spent working both on paper and on the computer. The group was divided bi-modally, with half having spent about 10% of their time on the computer, while the other half spent about 70% of their time on the computer.

Those who worked mostly on the computer believed they had compromised their design in relation to the software's restrictions. However, they were also more confident that their work methods and end product were similar to what they would have produced if they had worked on paper. In contrast, those who worked most on paper reported that they had extended their knowledge of the software more in order to implement their paper design. From this contrast in outcomes, Black concluded the students who worked mostly on the computer had not fully considered how much the computer had limited their solution.

Goel (1995) undertook a similar study where he asked a number of designers to create some design ideas for two similar scenarios. For one scenario, the designers used a computer drawing program (Mac Draw). For the other scenario, they used pen and paper. Consistently, the designers created a wider range of ideas using pen and paper. He was able to observe ideas evolving quickly and noted the occurrence of jumps from one idea to another when the designers worked with pens and paper. By contrast, when the designers were using the drawing program they tended to refine an existing idea rather than move from one idea to another. He concluded that the drawing program stifled the creative and explorative phases of the design process.

Bailey and Konstan (2003) created a computer sketch tool for the multimedia designer (Demais) that they evaluated against both traditional tools and a commercial design tool (Authorware). They found that designers ranked Demais the most useful for exploring behaviour, followed by pencil and paper and then Authorware. With Demais the designers explored the same number of ideas as they did with pencil and paper, and more than they did with Authorware (a similar result to Goel (1995)). Designers also ranked Demais the most useful for communicating behaviour to clients and Authorware least useful. Clients and designers ranked the pencil and paper design highest for creativity and Authorware the lowest.

Wong (1992), a user interface design practitioner, found with her design work that using computer produced diagrams too soon in the design process focused discussion and feedback on details such as 'should the button be 3D or flat' at a stage when the more pertinent issue may have been what functionality is the

button going to invoke. Her solution to this problem was to digitise hand-drawn sketches and then use a product such as MacroMind's Director™ to animate the sketch. Strothotte (1994) reports that some architects create their designs on a computer and then trace over them by hand in order to present a hand-drawn sketches to clients. The reasoning they give for this strategy is similar to Wong's (1992); they found that clients think a computer produced diagram is nearly finished and so are reluctant to suggest changes of direction. Squiggle (Insight Development Corp., 2000) is a software package that converts drawings from CAD software packages into a drawing that has a hand-drawn appearance.

Rudd et al. (1996) list the relative advantages and disadvantages they perceive in low and high-fidelity prototypes. They see low-fidelity prototypes as: cheaper, encouraging development of multiple design concepts, useful for communication, addressing screen layout issues, identifying market requirements. Their disadvantages are: limited error checking, poor detail, it is facilitator driven (this is in conflict with the opinions of Wagner (1990) and Wong (1992)), limited utility after requirements are established, limited usefulness for usability tests, navigation problems and flow limitations. In contrast, they claim the advantages of high-fidelity prototypes are: they are completely functional and fully interactive, have a clearly defined navigational scheme, and are useful for exploration, testing and marketing.

Designers clearly prefer low-fidelity sketching tools for the early stages of design. The arguments are:

- Hand-drawing is quicker (Goel, 1995, Stahovich, 1998 #36; Landay, 1996; Rettig, 1994; Wagner, 1990)
- Hand-drawing focuses attention at the appropriate level of detail (Bailey & Konstan, 2003; Goel, 1995; Gross, 1998; Tversky, 1999)
- Hand-drawing supports informality, ambiguity and abstraction (Goel, 1995; Goldschmidt, 1999; Gross, 1998)
- Hand-drawn designs elicit a wider range of feedback when collaborating with others (Wong, 1992)
- Hand-drawing stimulates designers to think more creatively (Bailey & Konstan, 2003; Goel, 1995)

When the design ideas are relatively stable the sketches are transferred into a digital form using discipline-specific software. A digital medium offers support that is not available with the low-fidelity tools. The standard computer editing functions such as cut, copy, paste, undo, redo, resizing and aligning are one of the great strengths of computer environments. The ability to 'save', 'save as' and send email attachments is so commonplace that we tend to overlook these facilities, such tasks are either difficult or impossible with paper or a whiteboard.

A computer design environment should offer the advantages of hand-drawn sketching and computer support for both editing and filing. In her 'Prototyping: A day in the life of an interface designer' (Wagner, 1990) described her ideal design environment as consisting of a large interactive whiteboard, a drawing tablet for herself and support for various levels of design refinement and emulation of interface functionality. In Section 2.03, we survey a range of sketching environments that have been developed.

## 2.02 Learning to Program

This project is about exploring the utility of a digital informal design environment concentrating specifically on the needs of novice programmers. This section, in order to incorporate the needs of programmers in general and novice programmers specifically, reviews both the literature on programmer construction by experts and the literature on learning to program.

The rate of change in computer programming is phenomenal; the first electronic computers that were built during World War II were programmed via plug boards, the 1950's and 1960's saw the emergence of the first commercial computers. These machines separated the programs into an operating system and application programs. The application programs were initially written in assembler but quickly moved to third generation programming languages such as Cobol or Fortran because they are quicker and easier to write. The 1980's saw the wide spread introduction of both personal computers (PCs) and database tools with built-in programming languages. By 1990's GUIs became the norm. We are now experiencing a diversification of computing devices with WAP phones, PDAs and

imbedded computers, each with its own unique interface and programming requirements.

With each new development programs have become smarter and programming languages have provided more built-in functionality. To illustrate, while a C program required about two hundred lines of code to create an empty form window under early versions of Microsoft Windows, current visual languages provide WYSIWYG form builders as part of the programming language. To create a new form from code, in the latter instance, would typically take only one or two instructions.

In some ways programming has become easier with each new generation of programming languages. However, Marsden & Thimbleby (1998) suggest that the form builders in current programming IDEs have been an afterthought and that there are inconsistencies between the form builder and programming language, and that this creates difficulties for learners. Also, object-oriented (OO) programming has increased the learning required (Robins et al., 2003), as students must learn not only the procedural nature of programming but also learn class and object structures. The distributed nature of control flow in OO programming makes this more difficult to learn (Détienne, 1990).

Basic programming is no longer the preserve of an elite few computer experts; university courses as diverse as engineering, business and geography typically include a compulsory course on computers and programming as most professionals are expected to be able to write their own macros for applications such as spreadsheets. Many of these introductory courses do not delve deeply into programming theory or OO techniques but rather encourage students to work with both GUI interfaces and event driven procedures.

Computer programming is a new discipline, and our understanding of how people program and learn to program is either drawn from empirical studies (for example Soloway, 1984) or from theories on how people solve problems such as those identified by Newell and Simon (1972). Subsection 2.02.1 describes both what is involved in the creation of programs and how expert programmers go about this task. Subsection 2.02.2 looks at research into learning and teaching

programming. Subsection 2.02.3 reviews models of the software development life cycle (SDLC) and discusses how user interface design is integrated into the process. The final part of this section, Subsection 2.02.4, examines the literature on user interface design strategies and hypothesises on how these strategies are likely to be beneficial when learning to program.

### 2.02.1 Programming

Programming is the task of analysing and solving problems using a computer programming language (Booth, 1990). It requires a wide range of knowledge and skills (Pennington & Grabowski, 1990). The programmer needs software design knowledge which consists of: problem deconstruction strategies, knowledge of basic algorithms and methods to construct a solution design. Knowledge of program construction is also required; program syntax, building, testing and debugging. Finally, independent of the knowledge of programming, the programmer needs knowledge of the problem domain (Prasad & Fielden, 2002).

Lammers (1996) interviewed 19 of the programmers who have shaped the computer industry. She asked them what programming *is*. They consistently describe programming as an art, a science and a skill, (for example Lammers, 1996 p.11, p.65, p.98, p.149 ). Educators generally describe programming differently; they state that to create a program the programmer needs both detailed knowledge of: the programming language syntax (skill), and the strategies for problem decomposition and algorithm construction (science) (Linn, 1985; Maheshwari, 1997; Shneiderman, 1980). They make no mention of art.

Lammers (1996) also asked many of the people she interviewed to describe *how* they created a program. Most replied that the first step is visualising the finished product. From there they progressively broke the problem down into smaller and smaller parts and then wrote the code.

*"The first step in programming is imagining - I like to imagine the structures that are being maintained, the structures that represent the reality I want to code ... The code for the most part writes its self, but it's the data structures I maintain that are the key. They come first and I keep them in my mind throughout the entire process." (Lammers, 1996 Charles Simonyi, page 15)*

*"You have to simulate in your mind how the program's going to work, and you have to have a complete grasp of how the various pieces of the program work together." (Lammers, 1996 Bill Gates, page 73).*

Others (Blum, 1992; Petre & Blackwell, 1999; Shneiderman, 1980; Soloway, 1984) have made similar observations about the way expert programmers undertake program construction. All these studies were of expert programmers who were in the main talking about large complex programs, such as operating systems and application packages that are written in lower level procedural languages, not applications programs constructed in current visual, object-oriented environments. Even so, there is a striking similarity between the process of creating a program and that of more artistic design disciplines (Petre & Blackwell, 1999).

Pennington (1987a; 1987b ), has examined how programmers understood programs others had written. She proposed programmers needed two kinds of knowledge about programs: the first, *text structure knowledge,* the knowledge of the control flow of the program; the second, *plan knowledge,* the knowledge of how the parts of the program go together. She also found that people who best understood the programs had a good understanding of both the program and the domain within which it resided, while people with a poorer understanding had understood either the program or the domain, but not both.

The task of creating programs has changed significantly in recent times. Procedural programming was dominant until the advent of GUI environments such as Apple Mac OS™ and Microsoft Window™. Most applications programming is now object-oriented and event-driven, and is written using programming languages and database tools such as Visual Basic™, Delphi™, C++, Oracle™, and Access™. The IDEs for these languages include a WYSIWYG form builder tool that allows the programmer to specify most of the

user interface visually rather than by code. GUI environments change the flow of control from programmer to user; with text interfaces and procedural languages the programmer decides when to request input and produce output, with GUI environments the user can enter information or click buttons in any order they wish, the program must deliberately disable controls to make them unavailable. In general, 50% of program code is directly related to rendering and controlling the interface (Myers & Rosson, 1992).

The move to GUI environments has been paralleled with an emphasis on object oriented (OO) programming. Procedural programs tended to separate data and code. With OO programs the design paradigm involves considering programs as a collection of objects which consist of data and related methods. While many have claimed that object-oriented programming is more natural and easier to comprehend, studies have not supported this. Risk (1996) suggests that OO programming "is more" to learn because of the overhead of the class structures.

Petre and Blackwell (1999) conducted a comparative study involving expert procedural programmers and programmers using a graphic programming language (LabView). LabView is a specialist language for displaying graphs that uses block diagrams to specify functionality rather than code statements. They found that the programmers used similar design strategies regardless of the programming language type.

Navarro et al. (2001; 1996; 1997; 1995), extended Pennington's work by exploring the differences in mental representation and comprehension, between procedural languages and visual languages, using C for the procedural language and a spreadsheet for the visual language. They found that there was a significant difference in the speed and type of understanding. After a quick inspection of a visual program the participants had a good understanding of both the control flow and the structural flow of the program. With the procedural program the understanding was just of the control flow. An understanding of the structural flow did occur when the programmers modified the procedural program, but the understanding was still less comprehensive than the understanding the visual program. They concluded that visualisation aids program comprehension.

### 2.02.2 Learning and Teaching Programming

There has been an ongoing interest in both teaching and the learning of programming, and as programming has changed so also have the challenges that novice programmers face. Programming draws on a diverse range of skills (Prasad & Fielden, 2002). From mathematics come the skills of problem solving, abstraction and deduction. From linguistics, the skills of language comprehension, reading and writing. From the creative arts, the ability to create and compose, both the user interface and the underlying program. From psychology, knowledge of how to construct a user interface that people find easy to use. Lastly, there is the domain knowledge, as it is very difficult to write a program without a good understanding of the problem the program is trying to solve. It is generally assumed that most tertiary students already possess these skills; however, it is quite likely that many are deficit in some areas (Prasad & Fielden).

Robins et al. (2003) provide a useful framework (Figure 6) of the skills and knowledge required by programmers at the different stages of program construction.

| | Knowledge | Strategies | Models |
|---|---|---|---|
| Design | of planning methods, algorithm design, formal methods | for planning, problem solving, designing algorithms | of problem domain, notional machine |
| Generation | of language libraries, environment/tools | for implementing algorithms, coding, accessing knowledge | of desired program |
| Evaluation | of debugging tools and methods | for testing, debugging, tracking/tracing, repair | of actual program |

**Figure 6: Programming Framework from Robins et al. (2003)**

In order to create a program the student programmer must be able to deconstruct and reconstruct problems (Barnes et al., 1997; Linn, 1985; Shneiderman, 1980).

This requires knowledge of the problem domain, of programming planning and of algorithm design. Students also need to learn the strategies that experts use to achieve this planning. Expert programmers hold chunks of knowledge and strategies together, which they then draw on, implementing each chunk as an input-process-output sequence. These chunks are the basis of software patterns. In contrast, students create a plan (sometimes) and then implement the process followed by the input and output (Rist, 1995).

Novice programmers often struggle with the program language syntax, yet ultimately this is the simplest part of programming. The rules of language syntax are well defined and the compiler directs the programmer to any transgression of these rules. However, for beginner programmers, particularly when they use syntactically complex languages such as C++, this can be a major hurdle.

To successfully design a program and to test it a programmer needs to consider both the complexity of the problem and the various paths of logic that may be required for a program to successfully complete a task. They must identify the range of values that the program may be required to deal with, and the different combinations of data and events that the program would have to handle. As a programmer becomes more experienced, he or she builds up a set of templates of standard algorithms and strategies for automating some of this process. But for the novice, there are often an overwhelming number of logic path combinations.

The last requirement of programming is to be able to mentally 'see' a solution, this is what designers refer to as imagery (Goldschmidt, 1991). Problems that are similar to other problems that the programmer has solved, maybe solved by template matching (Linn, 1985; Shneiderman, 1980). Many business problems are at this level in that they fit general business algorithms and patterns. Novice programmers do not have a catalogue of previous successful solutions to draw on, hence the necessity of the ten year apprenticeship from novice to expert (Robins et al., 2003). The creators of truly novel programs are clearly working at a higher level. These programmers have the ability to imagine new ways to use a computer to solve problems.

The traditional approach to the teaching programming has been to explore the programming language in small discrete sections and for the students to complete a number of well understood exercises to practise a particular skill (Fincher, 1999). In contrast, Linn and her colleagues (1985; 1992) have made extensive use of case studies to teach programming. They provide models of how experts solve programming problems for students to analyse. The students then solve problems following the experts' methods that they have explored. Milbrandt (1995) describes a similar method of focusing on specific problems and teaching towards a solution to these problems. Kay et al. (2000) have made extensive use of problem-based learning using Blue, a Java derivative.

Although there are anecdotal claims that it is easier to learn visual programming, exhaustive literature searches have found no studies to support this. Marsden (1997), who investigated the programming languages that exist with interface design tools, such as VB, suggested that they were not firmly based on good language constructs, and therefore, they were likely to have some detrimental effects on novice programmers. There have been many program visualisation systems which present abstract data structures or algorithms in a pictorial form. Mayer (1976) found that visual aids help students to understand Basic. More recently, Cañas et al. (1994) undertook a comparative study where one group of students learnt C in a standard environment while another group were learning with the support of a tracer program that visualised the internal computer memory and the sequence of program execution. They found that the students who learnt with the tracer had a better understanding of the semantics of the program.

Those interested in teaching programming to children, such as Papert (1980), have advocated the use of visualisations and constructive approaches, and have had considerable success with this approach. Many others have created tools for teaching programming, for example, Wright & Cockburn's (2002) tool includes the visualisations of objects and behaviours, and shows two representations of the program code, one is English-like and another similar to C++ or Java. They contend that the duel representation is likely to promote awareness of the specifics of standard programming languages.

As such a large portion of program code has to do with the user interface, we see working from the interface design as a useful way to explore problems and to develop programming knowledge, skills and strategies. A number of text books take this approach, yet this approach is not evident in the research on teaching and learning programming.

Another aspect of learning that may be relevant to learning to program is the learning cycle. A widely acknowledged specialist on constructivist learning cycles is Kolb (1984). He presented a four stage learning cycle (Figure 7) that consists of a concrete experience; reflective observations of that experience, abstract conceptualisation and further active experimentation.

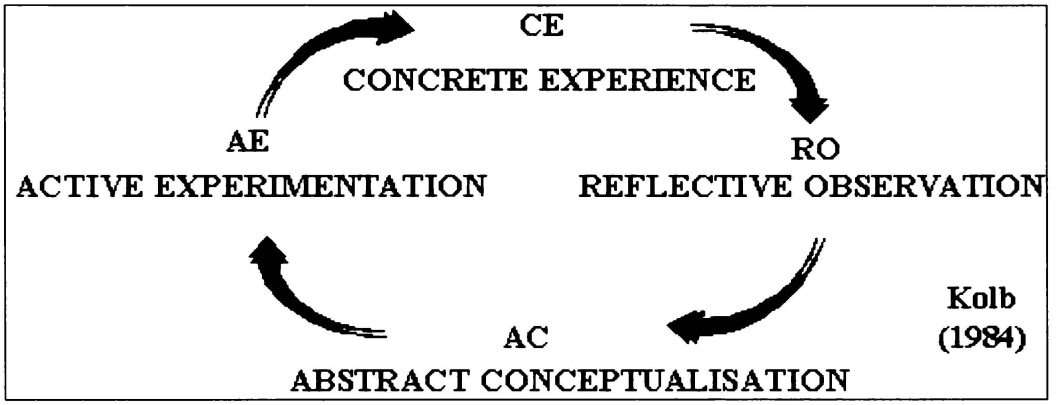



**Figure 7: Kolb's Learning CycleFrom (Greenaway, 2003)**

Kolb suggests that the more frequently learning can complete the more successful new learning experiences are. When novices are creating a program, there is often quite a time lapse between the examination of the program and the completion of its creation. Being able to check the user interface design, having some confidence that it is correct and that they understand the problem before starting on the program coding may have benefits for student programmers.

### 2.02.3 Software Development Life Cycle

There are a large number of theoretical models of the software development process. This section describes the classic waterfall model (Costabile, 2001; Johnson, 1992) , evolutionary prototyping (McConnell, 1996) and scenario-based design (Carroll, 1990, 1996, 2000), and considers where user interface design fits into each model.

The waterfall model stipulates a systematic, sequential approach to software engineering (Johnson, 1992). Various authors describe the process with more or less steps, but the general flow is: requirements definition, analysis, design, coding, testing and maintenance. User interface design is explored in the requirements definition stage and completed during the analysis stage. This model of software development infers a rigid partitioning of the stages; each stage being completed before the next is started. In practice this approach is rarely possible. Numerous variations on this model have been proposed incorporating feedback, however this defies the fact that the model is still based on a concept of separable tasks (Apperley & Duncan, 1994).

Costabile (2001) discusses where interface design and usability fit into the waterfall model. She suggests that to achieve user-centred systems user interface design must be better integrated into the software life cycle. At least, user task analysis should be undertaken as part of requirements specification, which is to say that both scenarios and user interface design are a critical part of both high-level and detailed system design.

Evolutionary prototyping, or rapid applications development (McConnell, 1996), is more iterative in that it is not expected that the first prototype produces the final system. The steps for RAD development are: initial concept, design and implement initial prototype, followed by iterative evaluate and refinement of the prototype until a satisfactory solution has been built. User interface design is central to this model of the SDLC. The initial prototype is likely to be little more than a user interface; as interface issues are resolved the functionality is developed.

Scenario based design (Carroll, 1990, 1996, 2000) also positioned the user interface as central to the SDLC. Carroll suggests that by considering use scenarios one can build up a description of requirements. This process involves a combination of both user interface design and use-case analysis, which he claims results in an understanding of the data, the underlying processes and the event responses.

Regardless of the SDLC model that is being followed, the importance of the user interface in modern systems cannot be ignored. From a user's perspective, the interface is the system (Apperley & Duncan, 1994). Myers (1994, p. 73) starts his article on the challenges of HCI with the statement that:

> *"Getting the user interface right is becoming critical to the success of products."*

There is a very loose connection between the theoretical models and the way experts program. What can be observed is that experts have a very clear mental image of what they are going to create and how the composite pieces are going to fit together before they start to program. Lammers' (1996) book also includes a number of sketches that are the original designs for some of the software the interviewees had written; from this material we can conclude that at least some of these expert programmers express their designs as sketches. What is even more interesting is that these men should still have their original rough sketches some years later; we can only assume that they continue to value these sketch artefacts.

### 2.02.4 Designing the User Interface

User interface design is a creative design task. All its essential characteristics are described in Section 2.01. Early design is also part of the software development life cycle. Computer user interface design is similar to engineering and architecture in that the products have both a visual and functional component. Yet unlike engineering and architectural designs, with user interface designs there is not necessarily a distinct separation between the design itself and construction. These other disciplines require a move from abstract design to concrete construction. User interface construction is infinitely more flexible than physical artefacts such as buildings or machines, as it is possible for the designer to construct and experiment with prototypes much more quickly.

Rettig (1994) proposes a design methodology where software developers begin by creating a paper prototype of the user interface, after which one of the team would 'play computer'. The interaction allows the team to focus on both the visual and functional requirements of the system. Rettig, like Carroll (1990; 1996; 2000) believes that rapid prototyping of the user interface, along with exploration of the

functional requirements, greatly increases problem understanding between the development team, while at the same time reducing both software development time and costs.

Wong (1992), in her paper 'Rough and ready prototypes', describes the same process from the perspective of a designer. She emphasises the same quick, low-tech prototyping as allowing design group participants to quickly define both system and interface requirements.

Wagner (1990) describes her idea of a perfect set of prototyping tools. She envisages a design team environment consisting of an electronic whiteboard that the team share and a drawing tablet for herself. During a design session, the team would sketch ideas and discuss possibilities. After the session, she would like to be able to transfer these sketch ideas into *visualisations*; sketches with some underlying scripting. She stresses that visualisations require very quick turn-around to be useful. The resulting interactive prototype would be still a sketch but some basic functionality allows the users to experiment with behaviour. From these prototypes further refinement produces prototypes for user testing and then final development.

Clearly by sketching the user interface and role playing its operation, designers can gain a better understanding of the problem, and the visual and behavioural requirements of the system. Also working from the interface encourages incremental and opportunistic development (Green, 1990).

### 2.02.5 Summary

This section provided a description of the way expert programmers create programs by first visualising the whole and then breaking down each part of the design until a detailed understanding of the problem is achieved. Only then does work begin on the construction of a solution. The research on learning to program suggests that student programmers need to learn the programming language syntax, problem deconstruction and reconstruction, and have sufficient domain knowledge to both understand a problem and visualise the solution. While learning language syntax may be difficult for beginners, ultimately it is solution

visualisation and problem deconstruction that are more challenging. A review of the software engineering process shows that user interface design is dominant in the early stages of both investigation and requirements specification. Finally, considering user interface design from a software engineering perspective, the research suggests that, by both rapidly prototyping the user interface and emulating the functional requirements of the interface, the user interface helps facilitate problem understanding and requirements.

What remains unknown is the application of early design techniques to the task of learning to program. We can conjecture that if sketching designs, rather than creating them with a computer-aided tool is preferable for expert designers, then it is likely to be preferable for beginner programmers too. It is also likely that beginner programmers will be better able to create their programs if they more fully understand the problem and solution space before they start to program, as this is how expert programmers work. It is also probable that by designing the user interface and thinking through the functional requirements of programs novice programmers will benefit from a better understanding the problem.

We would like to sound a note of caution here; we are not suggesting that user interface design or scenario-based design can completely replace more comprehensive systems analysis. Comprehensive systems analysis, depending on the methodology, includes exhaustive data, processing and object modelling. What we do believe is that simple user interface design and the functional analysis of requirements is likely to be sufficient for the types of programs that beginner programmers create in their first semester of study.

## 2.03 Computer-based Low-fidelity Design Tools

The previous sections have established that freehand sketching is the preferred visualisation method for early design ideas and that exploring the user interface in an informal environment is likely to be useful to programmers. Computer sketch tools are not commercially available because, from both a hardware and software perspective, it has been difficult to provide appropriate support. However, recent

releases of pen interfaces and increased knowledge of pattern recognition algorithms for intelligently interpreting virtual ink has seen a number of exploratory systems developed (Hearst, 1998).

The early work of Sutherland (1963) on cathode-ray screens is seen as a forerunner to pen interfaces. More recently the work of Xerox Labs (Elrod et al., 1992) on the Liveboard project and its use of this hardware for meeting support systems has triggered other research into informal interfaces. Sketch software has been developed for a range of different disciplines.

Three HCI sketch tools are reviewed. *Silk*, an environment for form design (Landay, 1996, 1998; Landay & Myers, 1995, 1996, 2001) is intended for designing interface forms. This sketch tool has the following facilities: a design space, a storyboard, a run mode and the ability to export the sketch in two alternate formats. *Denim* is based on Silk and is an environment for web page design (Lin et al., 2000; Newman & Landay, 2000; Newman et al., 2003). This sketch tool extends the views of *Silk* to five levels of zooms to show the structure of a web site. *Demais* (Bailey & Konstan, 2003; Bailey et al., 2001a, 2001b) is a tool for prototyping multimedia applications. This sketch tool places more emphasis on behaviour modelling and allows the designer to incorporate mixed-media clips, such as pictures, video or sound, into the sketch.

*Knight* is a tool for sketching CASE diagrams (Damm et al., 2000a), it permits an interesting mix of informal and formal widgets on the same diagram, and allows sketches to be translated to a formal CASE tool. Gross (1994) has developed a sketching platform that he has used as a basis for a number of different interfaces, such as for an index to web pages and for electrical diagrams. Stahovich et al. (1997; 1998; 1996) have analysed mechanical engineering diagrams to synthesise new designs that meet the same mechanical requirements. Trinder (1999) has worked with architectural drawing and has contributed ideas on the layering of sketches.

Each domain has its own set of unique requirements that depend largely on the properties of the final product. For example, computer interfaces such as a form, web page or CASE diagram are 2D so there is a simple match between the

dimensions of the sketch and the product. Architectural and engineering drawing depict 3D objects in 2D so the correlations are more complex. The functional requirements of a product depicted in an engineering drawing and in a web interface are vastly different, yet each can be emulated in a computer supported sketch environment. CASE diagrams are different again in that the diagram is an abstract model of a system where the final system is not a product that we can view or touch in the way we can a computer interface or a physical object. The end of this section draws together some general observations about the implementation of informal sketch interfaces.

This section reviews software that supports freehand sketching and focuses on five different aspects: the physical interface, sketch space capabilities, sketch recognition, the support for emulating functionality, and the transformation of the sketch into a formal environment.

### 2.03.1 The Physical Interface

In 1963, Sutherland used a cathode-ray screen and light pen to create an interactive drawing surface for his sketch-pad application. The phasing out of cathode-ray pens slowed the research into informal drawing. However, the recent advent of pen input and interactive interfaces has seen revived interest in this area of research.

Bly and Minneman (1990; 1991), at Xerox™ Palo Alto Research Center (PARC), have created a system that allows designers to work in real-time from different physical locations on the same design. At each site, they placed a monitor under a transparent digitising tablet. The tablet digitised stylus input to provide virtual ink. The sites are linked so that stylus input from each tablet is immediately displayed on the monitors at all sites. The software is based on a 'pad of paper' paradigm. The users can draw, write and flick through pages.

Xerox PARC also has developed Liveboard (Elrod et al., 1992) a digitised whiteboard. The image is back-projected onto a liquid crystal screen that measures 46 X 32 inches. Input is via a cordless pen which emits a beam of optical radiation when it is positioned near the board. The pen position is picked

up by a detector behind the screen and the xy position of the pen transmitted to the processor. The pen has three buttons and a tip-switch which allows for different modes of input. Liveboard is used as the hardware for Tivoli meeting support software (Moran et al., 1997; Moran et al., 1998; Pedersen et al., 1993) and Flatland an office whiteboard management software (Mynatt et al., 1999). These systems have not been commercially released. Liveboard is also being used in a classroom study by Abowd et. al (1996). Abowd et. al say that although Liveboard is expensive and too small for large groups to see, it provides an excellent interface for live capture in lectures. Ainsworth & Huddleston (1999) have similarly used Smartboards in classrooms. While they also comment on the cost, they find Smartboards to be a successful medium for demonstrating computer applications.

Wacom™ drawing tablets are the input device for *Silk* (Landay & Myers, 2001) and the electronic cocktail napkin (Do & Gross, 2001). These tablets gather both co-ordinate and pressure data. Its pen has a button on the side which can be used for different input modes. Gross (1994) made use of the pressure data as part of his recognition. While he also used two tablets connected to a single port to provide for two-pen input, only one pen can be used at a time. These systems used standard screens for output. The main disadvantage of drawing tablets is that there is an element of disconnectedness when the drawing surface is separated from the display (Gross & Do, 1996). Stahovich (1998) uses mouse input which, like drawing tablets, separates the input from its output. Additionally, most people also find drawing with a mouse difficult.

Other researchers have used interactive screens. Denim (Lin et al., 2000; Newman et al., 2003) was developed for use on tablet PCs. This research group has also used PDAs for associated note-taking work (Landay, 1999). Demais (Bailey et al., 2001a) has used graphical display tablets that allow the user to sketch directly onto the display. Knight (Damm et al., 2000a), uses a Smartboard (2000), a large interactive whiteboard. These devices have a distinct advantage over drawing tablets in that they allow the user to interact directly with the drawing surface in the same way as one does with a piece of paper. All of these devices support only single stylus input; the discriminating factor between them

being their size; PDAs are too small to be useful for any substantive design work, tablet PCs are likely to be useful for individual use, but difficult for a group to share. An interactive whiteboard provides a better setting for group work in that the group can share the space.

### 2.03.2 Sketch Space Capabilities

Central to sketching software is a space where sketches can be rendered and edited. The first requirement is to be able to draw freehand. Other facilities that have been provided are layering or a modal interface, zooming and radar windows for large space navigation, and general editing such as cut, copy and paste.

With Sutherland's (1963) sketchpad the user could create straight lines and arcs by first setting end points on the screen with the light pen and then setting switches on the control panel to indicate whether the line was straight or curved. Sutherland's work was forward-thinking and made good use of the computing power at the time. However, this method of design cannot be regarded as freehand sketching and, for this reason, no further reference will be made to it in this section.

More recent sketching applications support freehand inking. When the drawing device; pen, stylus or mouse, is 'on' the surface the device passes xy points to the program. The program then joins adjacent pairs of points with a straight line, the frequency of points means that the resulting line may appear as a continuous curve although it is constructed from straight line segments. Sketching applications deliberately focus on informal drawing and as such do not provide regular geometric shapes like line, rectangles and ellipses that are standard in other software.

Some of the applications provide a layered approach. Trinder (1999) has worked extensively on this idea in the context of architectural sketches. The paradigm involves progressive layers of tracing paper where the lower sketch layers are faded out as new virtual sheets are placed over them. Silk (Landay, 1996) has three drawing modes: drawing (strokes that are parsed by the recogniser), decoration (strokes that are not parsed by the recogniser) and annotation (notes

and glyphs that the user can add to augment the sketch). Annotation is shown in a different colour to drawing and decoration. Knight (Damm et al., 2000b), a CASE tool, has two drawing modes: freehand and UML, each being displayed in a different colour. Freehand drawing is not interpreted; UML drawing is immediately recognised as UML objects. Additionally, the user can select whether the recognised object is displayed unchanged, partially transformed or completed transformed (Figure 8).

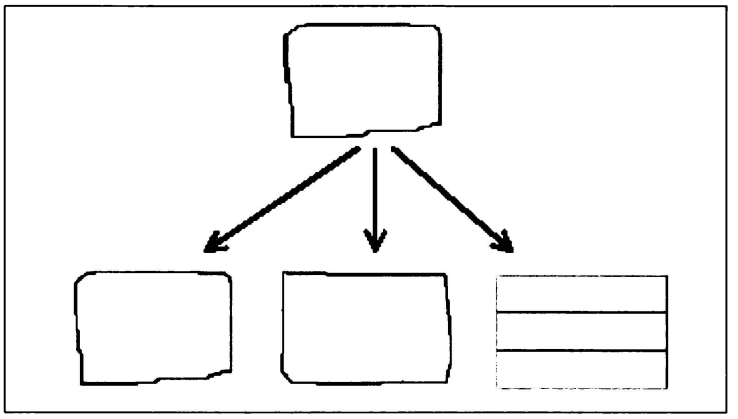


**Figure 8: Degrees of Transformation, From (Damm et al., 2000b)**

Denim (Lin et al., 2000) and Knight (Damm et al., 2000a) both provide zooming capabilities. Denim, a web site design tool provides five levels of zoom: overview, site map, storyboard, sketch and detail. Knight provides zooming and, also, because the workspace can be very large, a small radar window can be opened that shows the whole workspace in miniature with a rectangle indicating the section currently displayed.

Demais (Bailey et al., 2001a) is a multimedia design tool that focuses more on describing behaviour than sketching glyphs. To facilitate this, users can place pictures, video and sound clips onto the storyboard. Drawn rectangles are place-holders for these media clips. Inking can be used to annotate a sketch but emphasis is put on behavioural links. Joining two points can describe time-based behaviour, navigation links or narration synchronisation points.

Most applications support editing through gestures and/or context sensitive pie menus (Figure 9). Silk (Landay, 1996) recognises editing gestures when the pen button is depressed. There are gestures for deleting, grouping and ungrouping

glyphs, inserting typed text and cycling through gesture recognition. The electronic cocktail napkin (Gross, 1994), DocSketch (Pinto-Albuquerque et al., 2000) and Demais (Bailey et al., 2001b) take a similar approach to Silk. Denim (Lin et al., 2000) implements editing gestures and pie menus supplementing the Silk set with cut, copy, paste, move, pan and zoom. It also includes semantic zoom, when in the higher levels, overview and site map, a web page is treated as a single entity when at lower levels, glyphs or individual pen strokes are edited separately. Knight, in addition to pie menus (Figure 9) has short right and left gestures for undo and redo.

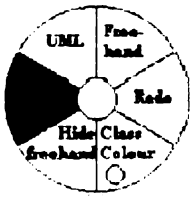



**Figure 9: Pie Menu, From (Damm et al., 2000a)**

In the sketch space most systems closely emulate the low-fidelity equivalent although even when pressure data is available, this data is not used in the rendering of the image. Layering has been used in two different ways: to separate ink marks that are intended to be recognised from those that are not (Damm et al., 2000a; Landay, 1996) and to imitate layered tracing paper (Trinder, 1999). Other media can be incorporated into a sketch in Demais (Bailey et al., 2001b). Zooming allows for navigation of a sketch space that is larger than a single display can show at one time and editing provides functionality that is not available on the low-fidelity equivalents.

### 2.03.3 Recognition

In order for a computer sketch application to be a useful partner in the design process it must be able to recognise sketch elements. Only by achieving this can an electronic environment participate in the next two parts of the design process: emulating functionality and transformation into a formal environment. Most computer applications have taken a two step approach: first, to recognise the

individual pen stroke as an instance of a particular shape and, second, to combine pen strokes into meaningful glyphs.

Stroke recognition has clear similarities to optical character recognition (OCR) with the advantage of having additional information relating to the order of creation of the stroke such as start point, direction of travel and timestamps. The most supported approach to stroke recognition has been Rubine's algorithm (Rubine, 1991), the reason being that it is simple to implement and train (Bailey et al., 2001a; Damm et al., 2000a; Landay, 1996; Lin et al., 2000).

Rubine's (1991) algorithm is a statistical single stroke recogniser and as such has two parts: training and classification. To create a classifier a training set is analysed using linear discrimination. The training set comprises subsets of example strokes for each class of stroke that is to be recognised (Figure 10).

From each example a number of features are extracted. Rubine (1991) settled on the extraction of thirteen features: the cosine and sine of the initial angle; the bounding box length and diagonal angle; the distance between first and last points, and cosine and sine of their angle; the total stroke length and total angles traversed; the sums of absolute angle of the angle at each mouse point and the squared angle of the angle at each mouse point; the maximum speed of a stroke between any two points and the total duration of a stroke.

These features are then averaged across a class and their weights computed to give a matrix of weightings to apply to input strokes. The result is a matrix of weighting values which can be used to classify new strokes.

To classify an input stroke the same thirteen features are extracted and evaluated against the weighting matrix. A value for each class of strokes in the library is calculated as the sum of the values of each input stroke feature value multiplied by the weighting for that class feature as computing from the training set. The smallest sum is the closest match. It is also possible to recognise ambiguous strokes and outliers.

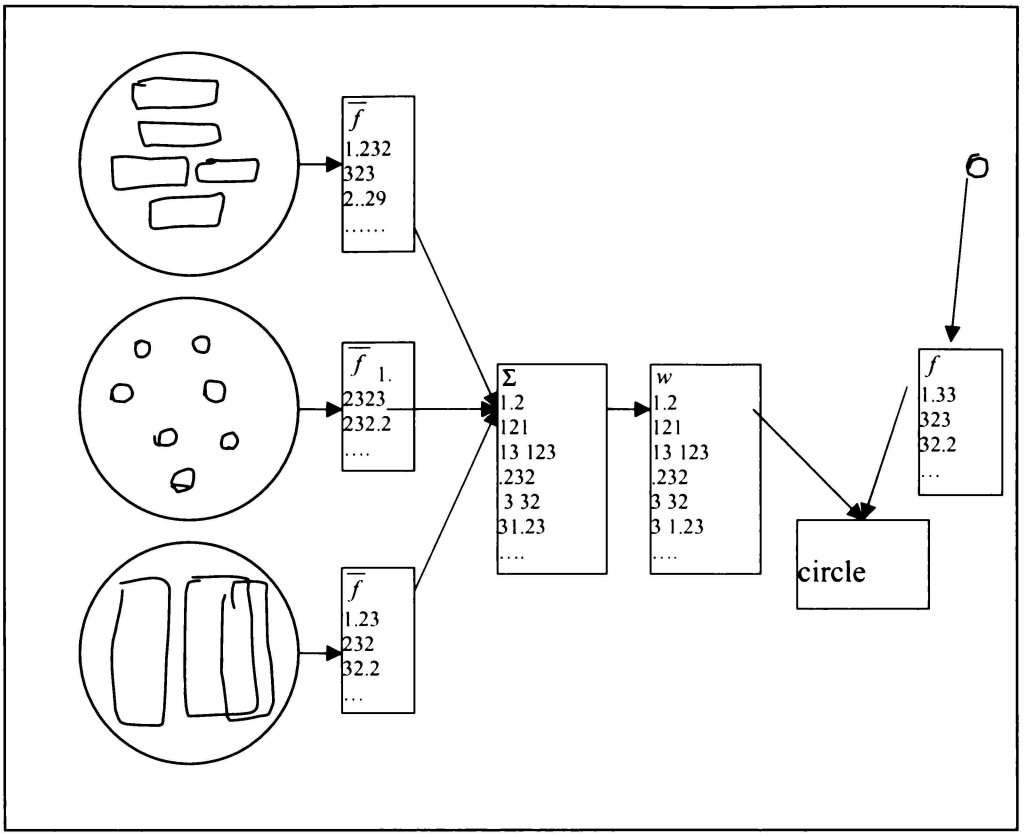



**Figure 10: Classifier Extracts Features From Sets of Each Specified Stoke Class and Uses These to Build the Classification Weightings**

This algorithm works best with a small number of geometrically distinct classes. Rubine obtained recognition rates of 98% with five and eight classes when forty examples of each stroke were supplied. This reduced to 96% for thirty examples. For most practical applications, fifteen to twenty examples of each class is sufficient.

Landay's implementation of this algorithm has been used for both Silk and Denim. He reports a recognition rate of 69% (Landay & Myers, 2001) which created problems when moving to the next stage of recognition. His team has also developed ideas on how to design strokes that result in better recognition (Long, 1998; Long et al., 1999). Damm et al. (2000a) and Bailey et al. (2001a) also used Rubine's algorithm.

Gross (1994) also used a statistical training set strategy but this strategy is based on overlaying the stroke with a three-by-three grid and counting corners. Rather than restricting users to a single pen stroke, Gross joins strokes when there is a minimal time delay between finishing one and starting the other, and when they

are close to each other geometrically. Pinto-Albuquerque et al. (2000) used a combination of geometric feature extraction and fuzzy logic.

The second part of recognition is undertaken once the individual pen strokes have been identified to build glyphs. For example, a button may be defined as a rectangle containing a text-holder (Figure 11). In order to do this the software must first recognise the spatial relationships between strokes: contains, is contained by, above, below, to the right or left. These relationships are then generally parsed through a rule-based algorithm to construct the most likely combinations. Landay (1996) has hard-coded these relationships. Gross (1994) permits the user to define new glyphs by example and then analyses the example to generate rules. The user can choose to make the rules more or less specific, for example, the user can choose between 'to-the-right-of' as being more specific than 'beside'.

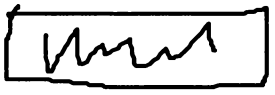


**Figure 11: Example of Two Stroke Glyph**

A similar approach may then be used to relate higher level components. DocSketch (2000) extends the recognition by applying fuzzy-logic to the relationships between parts of a web page design, calculating the likelihood of different relationships. For example, the software tries to identify headers and scroll bars, by identifying geometric attributes and relationships. Bailey et al. (2001a) also infers the existence relationships between objects depending on the source and destination object. In this case, such relationships are behavioural.

Stahovich et al. (1997; 1998; 1995; 1996) take a different approach to sketch recognition. Their goal is to take mechanical engineering sketches and recognise the mechanical interactions between the parts to construct a generalised working geometry. As such, they analyse the sketch to recognise the points where force and motion are transferred. The sketch is supplemented with at state-transition diagram that describes the required functionality.

The Design Rationale Group at MIT has recently published a number of papers on different aspects of sketch recognition. For example, Alvarado and Davis (2001) are reported to be working on disambiguating component parts of mechanical drawings and Hammond and Davis (2003) are reported to be working on language LADDER for drawing, editing and recognising shapes.

### 2.03.4 Emulating Behaviour

The underlying functionality that a sketch implies is domain specific. Silk (Landay, 1996) built up a storyboard analogy. The user can paste a sketch onto the storyboard and then draw a link from an element (usually a button) on one sketch to another sketch on the storyboard. When the software is in 'run' mode, clicking on the source element of a link opens the destination sketch. Landay also provided some limited behaviour emulation for screen elements, for example a user can drag a scroll bar handle up and down without affecting the scrolled pane. Denim (Lin et al., 2000) uses a similar approach, where from the site map the user can specify navigational links between page elements and pages. In run mode, link sources are shown in a different colour and clicking them displays the destination page.

Demais (Bailey et al., 2001a) places more emphasis on behaviour. In the sketch space, the designer can place links or markers to navigate, play, pause and end imbedded media, display, undisplay, highlight, unhighlight and synchronise audio with other media.

Gross (1998) has used his basic sketch environment as a front-end to a number of domain specific applications. Sketches have been used as indexes or bookmarks in these applications, linking information such as a URL or a record number that is attached to the sketch. The user can browse the sketches and click on a sketch to invoke the link. This platform has also been used as a sketch front-end to a network design tool and another tool which assists architects to calculate the visual territory from a particular position.

SketchIt's (Stahovich, 1998) purpose is quite different. By identifying the active surfaces in an engineering drawing and putting these surfaces together with a state

transition diagram, which the user provides to describe the required functionality, SketchIt can produce multiple new solutions to the problem. SketchIt first determines whether the sketch meets physical constraints and, if not the software corrects the input parameters (but does not alter the sketch). From its generalised understanding of the problem a family of solutions can be generated that all meet the engineering requirements of the problem although their parts may be arranged differently.

### 2.03.5 Transformation to a Formal Diagram

For every domain where computers are used for design there are many software packages available. Most of the work that has been undertaken with sketching has been aimed not at replacing these software packages, but at providing an informal interface with varying degrees of support for transformation into formal design environment.

Silk (Landay, 1998) outputs form designs in two formats: Visual Basic 5 code and Garnet User Interface Development Environment. Denim (Lin et al., 2000), WebStyler (based on Napkin) (Gross, 1998) and DocSketch (Pinto-Albuquerque et al., 2000) all produce HTML. Knight (Damm et al., 2000a) is integrated in a WithClass CASE tool.

This section has reviewed related work on computer sketching tools. Most of these tools use pen input; there being an evident trend towards direct interaction devices such as a tablet PCs or interactive whiteboards. These devices are preferable to a drawing tablet or mouse as they provide more natural interaction. Recognition, despite a number of well known techniques, is still difficult and, given the contradictory goals of providing an informal interface and accurate recognition, is likely to remain a challenge. Notably none of the systems discussed above have integrated character recognition. The possible ways that a sketch can imitate the behaviour of the sketch product is just beginning to be explored, as is the integration into formal design environments.

## 2.04 Summary

This chapter has reviewed the literature on early design, learning to program and computer-based sketch tools. Designers clearly prefer low-fidelity tools for early prototyping. The main advantages of these tools are that they are quick, and encourage ambiguity and exploration of ideas, whereas computer-based tools are slower and distract from the design task by requiring a selection of widgets and setting of properties such as position, fonts and colours. Evidence also suggests that hand-drawn designs are better for eliciting comment from team members. In contrast, computer created prototypes have the advantage of being able to demonstrate behaviour as well as being more useful in later stages of the design and construction process.

Programming is a complex task that requires a wide range of knowledge and skills. Because of this, it is difficult for people to learn to program as they must concurrently learn new ways to break down problems, programming language, and program construction strategies and techniques. As a programmer gains experience little disconnected bits of knowledge become chunked together allowing him/her to deal with more complex tasks but for novices, juggling, getting an understanding of the problem, and knowing the language and method to construct a program solution is a non-trivial task.

A number of other researchers have developed sketch tools for early design. Pen input was first mooted by Sutherland in 1963. More recently sketch tools have been developed for user interface design for form-based interfaces (Landay & Myers, 2001), web pages (Lin et al., 2000), multi-media designs (Bailey & Konstan, 2003), indexing (Gross & Do, 1996), document design (Pinto-Albuquerque et al., 2000), architecture (Trinder, 1999) and engineering drawings (Stahovich, 1997). Most of these systems accept pen input and render the ink in a drawing space. They generally incorporate stroke recognition to provide both functionality with in the drawing space such as cut, copy, paste and functionality for describing behaviour in run mode. Some also provide output to convert the sketch to a formal tool.

# Chapter 3
# The Conceptual Design of a Tool for User Interface Designs

Getting the design of the user interface correct before starting to write code is beneficial to novice programmers for the reason that if the interface is correct there is a good chance that he/she understands both the problem and required behaviour (Rettig, 1994). This chapter provides a conceptual description of a computer-based tool to support novice programmers when designing user interfaces. Firstly, to establish the high-level system requirements, the chapter draws together the elements of design, software development and the learning to program discussed in Chapter 2. Then, in the second part, we describe how these high-level system requirements may be achieved, given the constraints imposed by current technologies and assumptions, and the tradeoffs and refinements that we made in preparation for development of a prototype system.

## 3.01 Conceptual Description

The literature suggests four philosophical ideas that need to underpin a computer design tool: facilitating hand-drawn sketches, providing a shared group space, using sketches for exploring behavioural requirements and providing an environment that student programmers find attractive.

It is clear from the literature that working with low-fidelity designs has advantages at the early stages of problem exploration (Bailey et al., 2001b; Goel, 1995; Wagner, 1990; Wong, 1992). Sketching must be fast and minimally constrained by the tools. This implies that an interface that closely emulates that of the pen and paper/whiteboard is likely to be most natural for users. We

consider that the interaction should be restricted to direct pen input onto the surface that displays the ink output.

The literature also asserts that working in groups results in both better designs and a better understanding by group members (Bekker, 1993; Bly & Minneman, 1990; Bødker et al., 2001; Tversky, 1999; Wong, 1992). Therefore, we propose a large shared space for design such as a digital-whiteboard.

To understand the functional requirements of a program it is important to be able to interact with a design while it is still rendered as a sketch (Wong, 1992). The design literature asserts that reviewing a sketch is preferable to reviewing a formal diagram because there is less commitment to the sketch if it does not look finished, and it therefore seems easier to change (Goel, 1995; Wong, 1992). The literature also states that it is usual to explore behavioural requirements alongside aesthetics during this first informal stage of design. It is appropriate to encourage this for user interface design as scenario-based and user-centred design methodologies claim that by 'playing computer' (Rettig, 1994) with informal sketches of the user interface, the design team is likely to gain a better understanding of the problem domain (Cañas et al., 1994; Carroll, 2000; Rettig, 1994).

These ideas lead to the requirement of including a way to 'play computer' with designs before they are actually formalised. It is not clear how this should be best done. Rettig (1994) suggests using paper prototypes and sticky note pads to pass data around the prototype. Others have used computer tools to provide basic navigation and behaviour (Landay, 1996; Wong, 1992). Given what is known about the adverse affect of formalising a diagram too soon, it may be that the provision of too much behaviour emulation is also detrimental. Therefore, we propose a minimum of functionality for the play mode. In keeping with what is available on whiteboard and paper, we suggest that, early in the design process, the ability to view and 'fill in' a design, and navigate from form to form is all that is appropriate.

Finally, current IDEs tempt students to design with the IDE form builder. They see the support for editing, copying and storing designs in a computer

environment as superior when one thinks of these facilities in relation to paper or whiteboards. To encourage student use, a computer-based low-fidelity tool must provide these facilities. Also they see low-fidelity designs as an unimportant, time-wasting step because the artefact is thrown away. Landay's (1996) experiments suggest that it maybe quicker to create an interface design with a sketch tool. We aim to create a low-fidelity computer-based design tool that allows students to move freely along the continuum from informal, ambiguous sketches to formal interface designs, without the need for them to manually recreate the formal design from the sketch.

In order to achieve this, the computer tool needs to be able to recognise, with minimal effort from the user, the various elements of the design and convert them into a formal diagram in the IDE form designer.

To summarise the concepts that are important in the development of a sketch tool are:

- to retain the informal, direct, unconstrained nature of pen and paper/whiteboard for the creation of hand-drawn diagrams
- to provide a drawing space that is large enough for a group to share
- to provide support for editing and storage of sketches
- to include a way for users to emulate the use of their design sketches
- to integrate the software with the formal design environments used at the next stage of software development

There are some conflicts when it comes to being able to realise these requirements as functional components of a real system. In particular, at the present time it is unlikely that a rough sketch, which does not adhere to any rules, would be able to be recognised by software. However, a number of sketch tools have been developed and these provide some guidance to what may be possible. In particular, Landay and Myers' (2001) tool, Silk, provides ideas on both the sketch space capabilities and the run mode. We would like to extend their ideas and integrate the sketch design space into an IDE and include handwriting recognition,

so that users could continue to work exclusively with a pen rather than having to switch to a keyboard when adding words to their sketch. Additionally, expose the recognition libraries and rules to the users so the software becomes more extensible.

## 3.02 Refining the Requirements

This section describes the ideal tool based on the preceding requirements and the technology available, in 2000, when the work began. We planned to use the LIDS (Apperley et al., 2001; Apperley et al., 2002) hardware and hoped to be able to use commercial character recognition components. While there are a number of IDEs in common use, we have chosen to integrate our tool into Microsoft Visual Basic 6©, as it is often used as a medium for teaching novice programmers. The following sections provide the details that characterise the initial development of our first prototype.

### 3.02.1 Physical Interface

We wish to provide a shared workspace where the physical interface is based on a whiteboard paradigm which is simple and unobtrusive. We envisage the most successful implementation to be a digital whiteboard using transmitting pens. The transmitting pens should be a similar size to standard whiteboard pens (Figure 12).

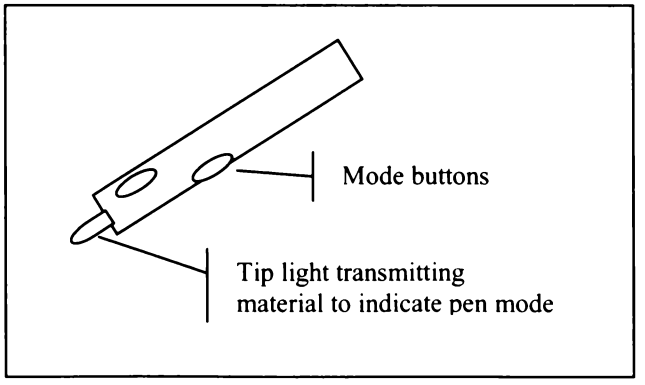



**Figure 12: Pen Design**

When the pen is close to the board, movement is passively tracked with a visible cursor. When contact is made with the board, the pen input creates digital ink. The pen buttons are used to change the function of the pen. These functions should be pen, editing tool and eraser. The pen indicates its current mode by changing the colour of the tip via small internal lights.

Pen input immediately updates the image on the digital whiteboard so that, except for the editing capabilities, there is little difference between the digital experience and that of using a normal whiteboard.

As hardware such as described above is not available, we plan to commence our project by using LIDS (Apperley et al., 2001), which provides an interactive whiteboard where the pen input is restricted to a single tip switch, which is activated when the pen is in contact with the board.

### 3.02.2 Sketch Space

One of the main advantages of sketching early design is that the cognitive load associated with the sketching process itself is minimal, meaning the major part of the cognitive effort can be directed towards the design process (Goel, 1995; Goldschmidt, 1991). The challenge with computer supported sketching is to honour the traditions of sketching while providing the support expected of a computer tool.

We envisage an unconstrained drawing space that also provides the type of editing facilities that are usually associated with computer applications. Two views of the drawing space are provided: a single sketch view and a storyboard. The single sketch view is the main drawing space where the user draws, edits and annotates a form representation. The storyboard view shows all forms in miniature, as such allowing both the forms to be manipulated as whole items and relationships to be established between them.

The capabilities that a pen-based sketch environment should offer are not well established. Should an application provide a number of layers, as Trinder (1999) proposed, or should it provide modes as Landay (1996) proposed? What editing functions are required? Should editing be supported by gestures, menus or both?

Some decisions will need to be based on hardware and software constraints, for example, the Smartboard (2000) and Mimio (1999) pens do not have a button for mode changes, while the recognition of pen gestures is difficult and error prone. These issues are further explored in the prototype system.

Other researchers (Landay, 1995; Lin et al., 2000) have integrated the drawing and editing modes but have not included handwriting which further complicates recognition. Bailey and Konstan (2003) noted users requests for character recognition in their evaluation study. We believe that, to be truly useful, character recognition is necessary and we therefore plan to include this feature. Although, ideally, we would like to achieve a modeless interface where users can draw, write and edit freely, we have resolved to initially use three modes: drawing, writing and editing to reduce the problems of recognition.

A storyboard view that shows the set of sketches in miniature, similar to that of Silk (Landay & Myers, 2001), is also planned. On the storyboard designs will be able to be moved around or deleted and navigation links will be able to be created between forms that can be used in the sketch run mode. The metaphor here is of sticking paper sketches on a whiteboard and drawing links between them.

### 3.02.3 Recognition

Recognition is important to support editing gestures with the pen, for example, the delete gesture, and to automatically transform a sketch to a formal diagram. Sketch recognition is discussed in this section along with the vexed question of when to disclose recognition. There are three aspects to recognition: the recognition of a single stroke, the combining of strokes into glyphs, and the recognition of words as distinct from picture glyphs.

Ink recognition is dependent upon the ability to be able to pattern-match pen strokes to a predefined set of symbols. As individuals draw and write in different ways it is helpful if the system is able to learn the idiosyncratic writing/drawing style of each user. There are two ways to achieve this: by automatically updating the recognition data whenever the user corrects recognition, or by giving the user

access to the recognition algorithm's training sets to add and remove examples when they feel it is necessary.

Our casual observations of people sketching suggest that the informal nature of a sketch means that users draw ill-formed strokes that should not alter the recognition data. We do not wish to discourage this, as ambiguity is one of the essential elements of informal interfaces (Goel, 1995; Goldschmidt, 1991; Tversky, 1999).

As this software is intended for experienced computer users, it is anticipated that they would prefer to have control of the training sets. We plan to expose the training sets to the user and allow him/her to define whatever new class of strokes they think would be beneficial to their design work. The training interface will also include a test option that indicates any training stroke that falls outside acceptable variations. In the instance when an outlier occurs the user is able to add further examples similar to the miss-recognised stroke, remove the miss-recognised stroke from the set, or ignore it.

Glyphs may be similarly defined by example (Gross, 1994). A glyph may be a single stroke or word with no relationship to other strokes (e.g. a line), or a more complex arrangement of strokes (Figure 13).

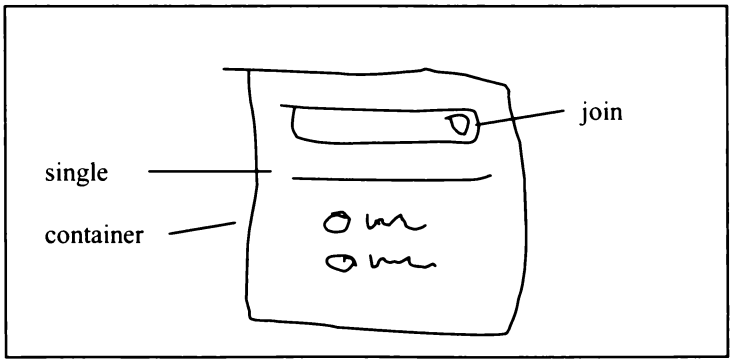



**Figure 13: Glyph Types**

There are two types of relationships between strokes that are of interest: joins and containers. Joins describe two or more strokes that combine to make one glyph (e.g. dropdown boxes). Containers, such as frames, are important in VB as option buttons within each container form a mutually exclusive set. Words can be treated as a single stroke either by being transformed into the caption of another

control, as defined by a *join* (for example, button caption), or by being transformed into a VB label.

User feedback of recognition is a complex issue; we intend not to provide continuous feedback. This is contrary to the situation of using most other sketch recognition software (Bailey & Konstan, 2003; Damm et al., 2000a; Gross, 1994; Landay, 1996). Recognition is required only for sketch run mode or transformation. Prior to this point the sketch can contain abstract or ambiguous elements. Our hypothesis is that providing immediate feedback on recognition may distract the designer from the design task at a time when recognition may be unimportant; we do not want concerns about recognition to distract the designer from the informality and ambiguity that is so important to early design work.

We sum up our approach to recognition in the following way: although recognition is technically challenging, it is thought of as a means to an end, not an end in itself. Recognition allows the sketch to react in run mode and be transformed into a formal environment. Algorithmic recognition is not 100% accurate and, as such, is not likely to be in the foreseeable future. Disclosing recognition while the user is sketching is likely to distract him/her from the primary task of design. A better approach is to provide a distinct *recognised* view of a sketch that has the ability to easily alter any miss-recognition. To allow for both personal differences and the expansion of the VB control set, the user is able to access both the training sets and rules for recognition purposes.

### 3.02.4 Run mode

From a programming perspective, while the interface design is often indicative of the underlying data structures, it does not illustrate the required behaviour. Research has shown that while designers consider functionality when they are creating a design (Bailey & Konstan, 2003; Goel, 1995; Gross, 1994), this in itself is not sufficient to uncover the functional requirements. Rettig (1994) recommends playing computer with a paper prototype as a way of responding to this situation. Wong (1992), for her part, has described both building low-fidelity computer prototypes by scanning sketches into a tool and adding navigation links as the means by which one can respond to this same situation.

Landay (1996) provides a run mode for the computer sketch and our plan is to take a similar approach. When the user changes to run mode it is as if a clear overlay has been placed over the sketch meaning the user can draw on this overlay without it changing the underlying ink. A recognised sketch could return substantial feedback. For example, such recognition would be capable of emulating the interaction of a set of radio buttons. However, because it is not clear that this is useful, when working on the overlay, we plan instead to supply only navigation links to other forms.

### 3.02.5 Transformation

The final facility that we suggest would be useful in a sketch environment is the ability to move freely between the informal sketch and a formal design. We plan to use a tightly integrated environment where the sketching system becomes part of the programming IDE. The user could take a finished sketch and transform it into the programming environment form builder.

As indicated above, it is at the point of transformation that sketch recognition becomes important. We propose a two-stage sketch-to-form transformation. First, the recognition engine parses the sketch elements, overlaying the sketch with a graphic and a label indicating its VB control equivalent. At this point, the user can correct any miss-recognised glyphs. When the user is satisfied with that recognition has been achieved, he/she instructs the system to create the forms.

The software also allows the user to describe how the properties of the VB control are to be generated. For example, the user can specify that the position properties of a textbox be generated from the glyph position (e.g. most left point of glyph => left property of VB control).

Once the VB form is generated, the standard IDE form builder becomes the active window. If the user re-enters the sketch environment from the IDE form builder the system checks for any changes on the forms and, if necessary, alters the sketches to reflect these changes. In the same way, the user can take a form created in the form builder and transform it into a sketch, then alter it and transform it back into a form.

Our system being intended for student programmers, we see integration and transformation as important features in encouraging students to undertake informal design. The reason these features are important is that the excuse students most frequently give for not sketching designs is that they see sketches as 'waste of time'. The sketch system outlined in this chapter would allow students to create a sketch in an environment where there is the support they expect of computer tools. This would mean they could then automatically carry this work through to the next stage of program construction.

This chapter described both the concepts that are important for the performance of our sketch tool and how we anticipate implementing these concepts within a real system. As much of this work is exploratory we have chosen a prototyping approach to systems development. The following four chapters describe (1) two iterations of prototyping and (2) the usability studies that have been completed in relation to these iterations.

# Chapter 4

# A Prototype Interface Design Environment

This chapter describes the first Freeform prototype as developed in 2000-2001. When implemented, this prototype provided a basic sketch-space, a recognition engine and the sketch-to-VB form transformation, meaning we were able to confirm the technical feasibility of the project and had a platform for an exploratory usability test.

This system runs as a Visual Basic add-in and interacts with the Visual Basic form designer. The program is written in VB 6.0 and can be run as a part of any standard VB 6.0 installation. A brief explanation of the VB6 IDE is given in Section 4.01.

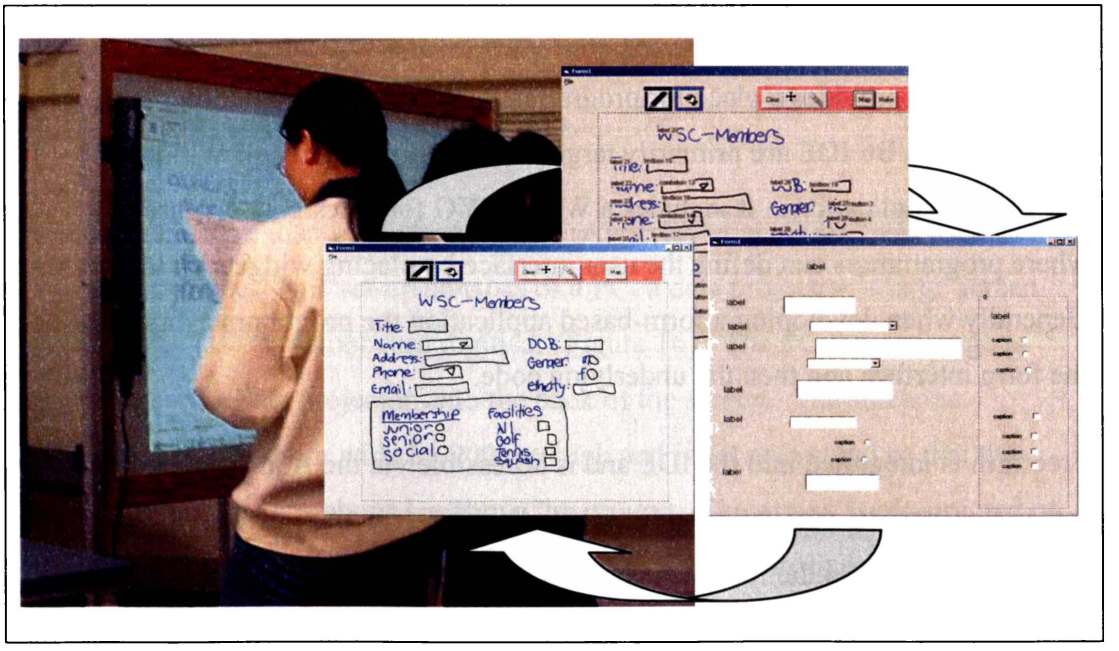



**Figure 14: Design Process**

The process for creating a design using Freeform (Figure 14) is as follows: a small group of designers use both the digital whiteboard and the Freeform software to

create a low-fidelity design. When the designers are satisfied with what they have assembled, the recognition engine overlays the design with the component names and corrections are made. The sketch and recognition data is then used to generate the VB form.

The system development is described using the same framework as was used in the preceding chapter. Section 4.02 describes the physical interface. This description is followed by a report on the sketch-space capabilities (4.03) and the recognition engine (4.04). Sketch run mode is not implemented in this prototype. Section 4.05 describes the transformation from a sketch to VB form. Section 4.06 summarises what has been achieved with this prototype. During the development, a number of informal evaluations were carried out where other members of the research group were asked to test particular components of the software. The next chapter describes the usability testing of this prototype and the subsequent enhancements which were planned for the next prototype.

## 4.01 Visual Basic 6 Integrated Development Environment

The VB6 IDE provides a place for programmers to create and test their programs. VB6 and the VB6 IDE are primarily targeted at creating Microsoft Windows™ form-based applications. It includes a WYSIWYG form designer (Figure 15) where programmers can define the user interface by placing widgets on to a form. Generally when developing a form-based application the programmer first creates the form interface and then the underlying code.

Freeform is integrated into the IDE and is accessible via the Add-Ins menu. The integration posed a number of technical challenges. In order to create form widgets on a form in the form designer window the Freeform program had to run as a compiled DLL. The widget creation process so is deeply imbedded within the IDE that errors in the program that creates the widget often resulted in parts of the IDE being destroyed, meaning a reinstall of VB6 was required to repair the damage.

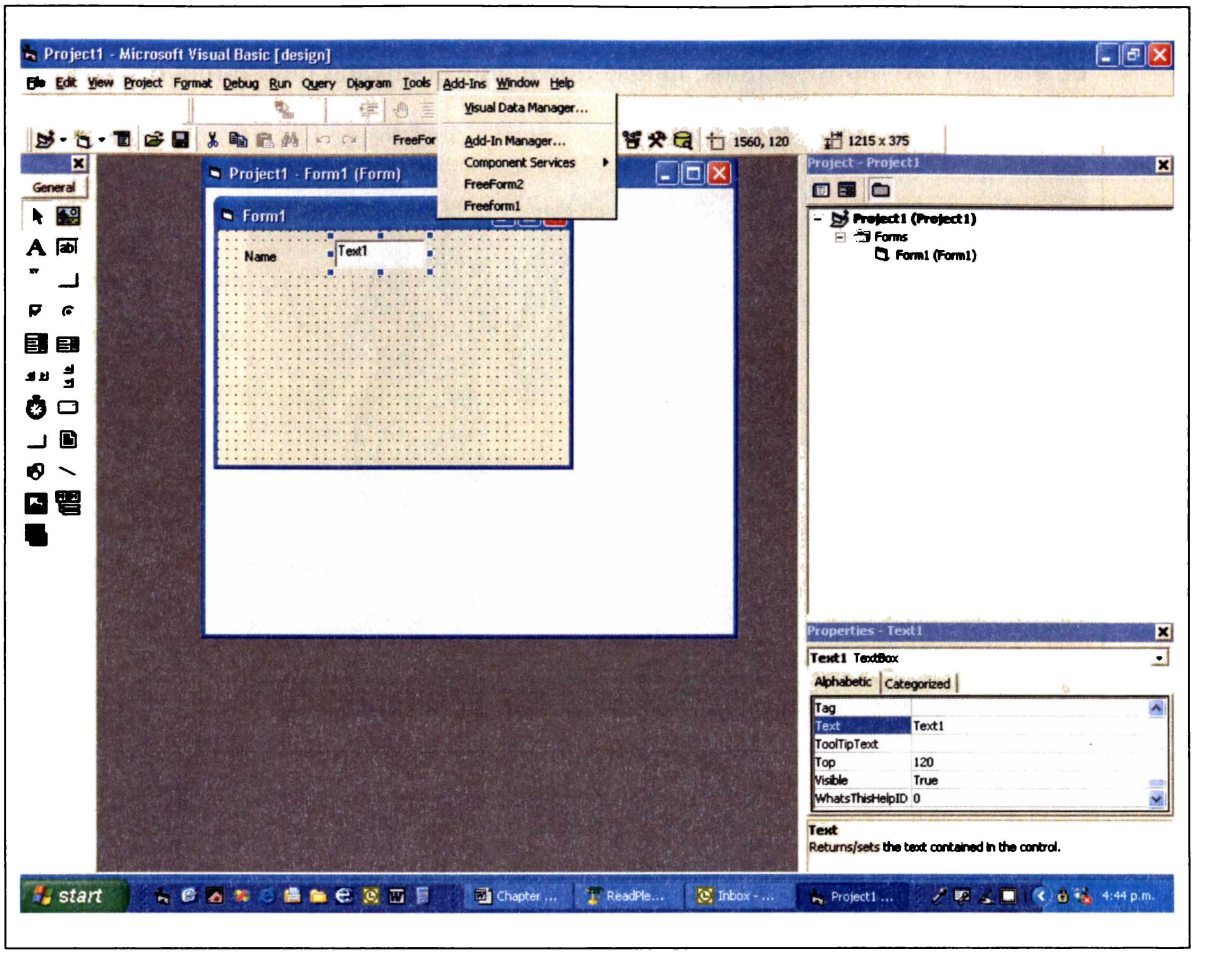



**Figure 15: VB6 Form Designer Window**

## 4.02 Physical Interface

A large interactive display surface (LIDS) (Apperley et al., 2001) was used as the physical interface. This set-up consists of a PC, a data projector, a large screen and a Mimio (1999) whiteboard digitiser (Figure 16). The PC screen image is projected by the data projector onto the back of the screen. The screen is approximately 1200mm wide by 900mm high and is set on a stand so that the bottom of the screen is about 1m above the ground. The screen is constructed of two sheets of clear glass that have been joined together with an opaque resin; this provides a durable flat surface with enough opaqueness to hold an image.

A Mimio whiteboard digitiser bar is attached to the side of the screen and used in mouse emulation mode. The Mimio pens transmit a signal to the digitiser bar

when the pen tip is depressed. This signal is received by the software in the form of standard left mouse actions; mouse-down, mouse-up and mouse-move.

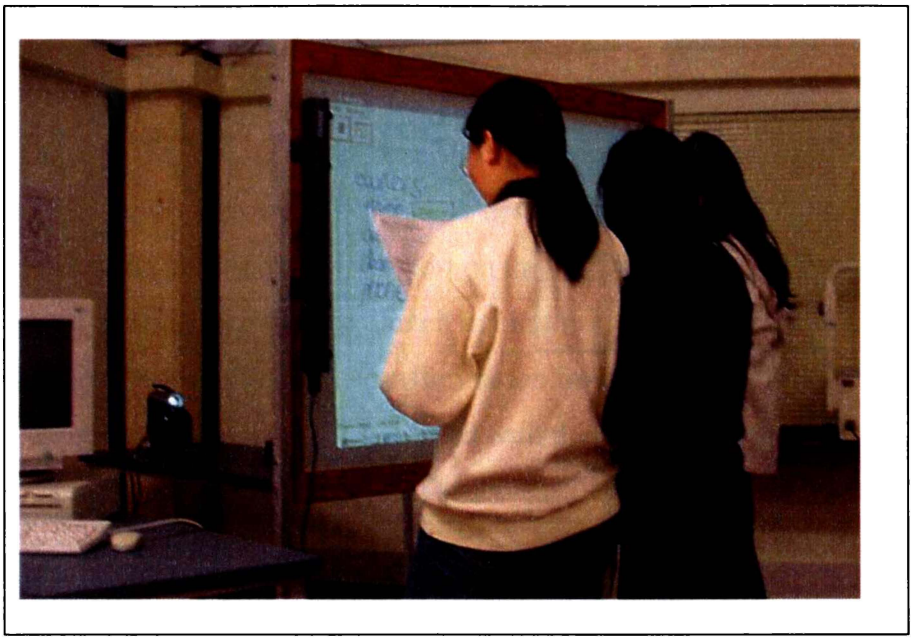


**Figure 16: Lids setup**

## 4.03 Sketch-Space

The sketch-space (Figure 17) is based on a whiteboard paradigm. It provides freehand inking and computer supported editing while at the same time sustaining an interaction that keeps the design activity as simple and intuitive as possible.

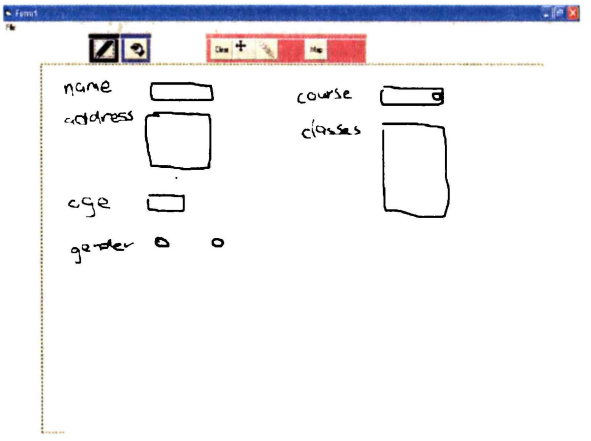



**Figure 17: Sketch-space**

A sketch can be saved or loaded from disk via the file menu. Most other sketch environments have a number of different modes, which are typically distinguished as drawing, editing and annotating (Bailey et al., 2001a; Gross & Do, 1996; Landay, 1996).

Initially, we wished to avoid the modal approach but early trials suggested that this tactic was unlikely to be successful because of both the inherent inaccuracy of recognition and the user's low-level of confidence when using gestures to invoke functions. The resulting sketch-space has four modes: drawing, writing, editing and mapping. These modes are selected via icons along the top of the screen (Table 1).

| | Draw | Write | Edit | | Map |
|---|---|---|---|---|---|
| | | | Move/ Resize | Erase | |
| Icon |  |  |  |  |  |
| Cursor |  |  |  |  |  |

Table 1 Mode Icons and Cursors

Each mode is visually different. In the drawing and writing mode, the sketch is shown as drawn (Figure 17) with both different pen cursors (Table 1) and ink colours for each mode. The user can select the pen colours from a set of colours that are easily seen on the LIDS display (Figure 18). In edit mode, each stroke and word is surrounded by a dotted-line bounding box (Figure 19). In map mode, the sketch is overlaid with the recognition information (Figure 20). The following sections describe each mode in detail.

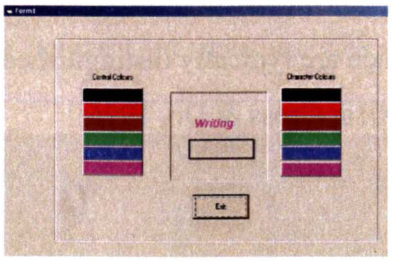



**Figure 18 Colour Selection**

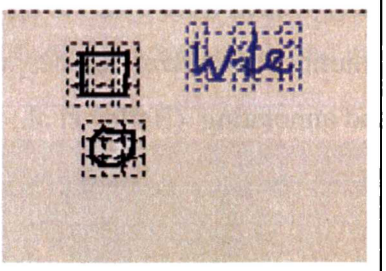


**Figure 19 Edit Mode**

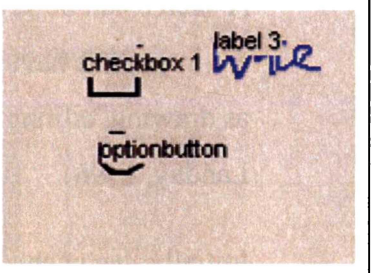



**Figure 20 Map Mode**

### 4.03.1 Drawing Mode

Drawing mode is used for the freehand sketching of glyphs. The user may draw anywhere on the sketch at anytime. During development we found that some users wanted to draw a border, which in turn led to recognition and transformation problems. To overcome this we introduced the dotted line-border, which as a design affect is only cosmetic (Figure 17). The user can resize the border and draw inside or outside the line.

The software accepts mouse events from the pen. Each pen action results in a sequence of mouse events beginning with a mouse-down, followed by a number of mouse-moves and ending with a mouse-up. The mouse-down event initialises the creation of a new stroke object which contains both the start position and a collection of time stamped offset points. Each mouse-move event is passed to the add-point method; the first three points are added without alteration (something required for the recognition algorithm). Subsequent points are checked and those lying on a straight line are discarded to minimise the number of points held (Figure 21). As points are added to the stroke virtual ink is displayed along the pen path.

At the completion of a stroke (mouse-up) the pen path information is passed to the first phase of the recognition engine. Two editing conditions are identified in draw mode: erase and overdraw.

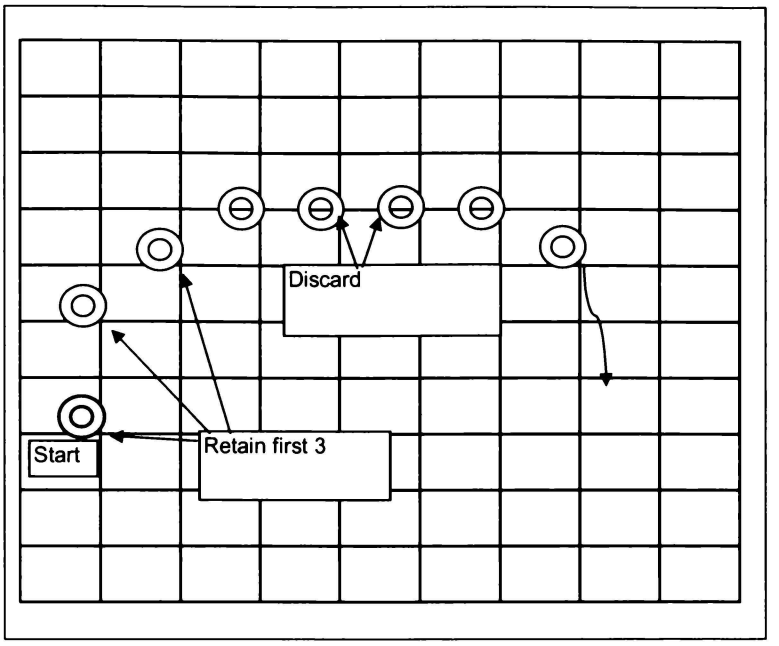



**Figure 21: Mouse Event Points**

If an erase gesture (Figure 22a) is placed over a previously drawn stroke(s), so that its bounding box covers the old stroke(s), both the erase gesture and the underlying stroke(s) are removed. In the case of the other condition – overdraw– if one stroke is drawn over another so that their bounding boxes are approximately the same size then the underlying stroke is removed (Figure 22b).

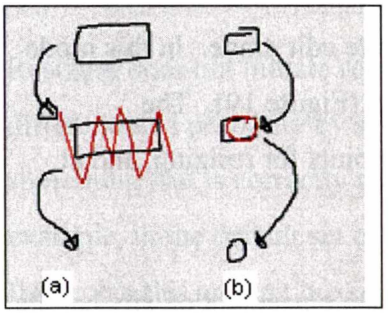



**Figure 22: Editing in Draw Mode**

### 4.03.2 Writing Mode

Writing mode is similar to drawing mode in that pen strokes are free form. However, letters adjacent to each other are grouped into words (Figure 23). Adjacent measures were arrived at by trial and error. Just as letters that are grouped cannot be ungrouped, groups cannot be joined by moving two groups closer together. However, extra letters can be added to a group at any time by

simply writing another letter close to an existing word. As with drawing mode, each stroke is passed to the first phase of the recognition engine, which leads to erase being both immediately identified and acted on.

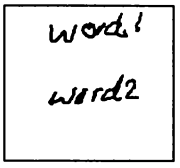



**Figure 23: Letter Clustering for Words**

### 4.03.3 Edit Mode

Editing proved to be difficult to provide in an intuitive, user friendly manner. The main reason is that LIDS does not have passive pen tracking. The initial plan was to have only erase and single stroke or word move, all actions which we felt could be achieved with gestures. However, testing during development indicated that functional gestures were likely to cause difficulty. Some users suggested that editing gestures were not natural and if the gesture was not recognised they then had more work to correct the sketch. Also, it was quickly apparent that people usually draw in too large a manner on a whiteboard and that resizing is frequently required. As well as the delete gesture and the editing condition overdraw described above, it was decided to incorporate a separate edit mode. In this mode each stroke and word is displayed with a bounding box (Figure 19). The bounding box has handles at the corners and side midpoints for resizing, and at the centre for moving.

The first approach to ink selection involved having a pen placement on any handle that would activate the appropriate resize or move function, or alternatively the user could select a group of strokes by starting outside any stroke and lassoing the group. The lasso stroke's bounding box would bind the group, which could then be resized or moved together. However, a small trial proved this strategy to be unsatisfactory, as users were confused about what was selected and whether it was going to be resized or moved. We changed to a two-step process. In this process, first, the user selects a stroke or group of strokes. A single stroke is selected by tapping inside its bounding box and a group is selected by a lassoing stroke.

Figure 24 shows screen shots of: stroke (a), word (b) and group (c) selections. The software responds by highlighting the selection's bounding box, after which the user is in a position to resize or move the selection. If a group selection is chosen the bounding boxes are removed from items that are included in the group. An erase button (Table 1) is provided so that users can erase by selecting and clicking the eraser.

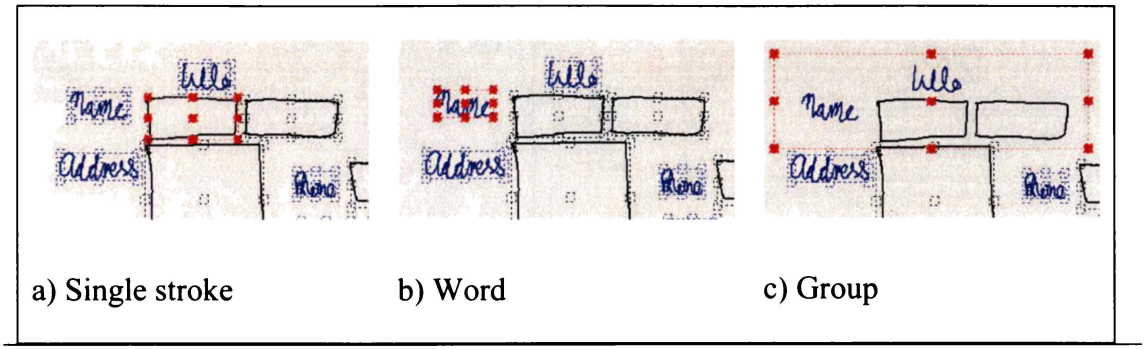



**Figure 24: Selected Ink**

We struggled with processor power when redrawing. If a large section of sketch is moved or resized, responding to every mouse-move event results in rather poor response rates. We compromised by ignoring many of the mouse-move events. The result involves slightly jerky resize/move operations that do not lag too far behind user actions.

Resizing does not initiate re-recognition even though some of the strokes are differentiated primarily by size. This allows a knowledgeable user to draw something that is correctly recognised and then resize it to the desired size. For example, in the default set of shapes, small squares are checkboxes, slightly larger flat rectangles are text boxes and larger rectangles are list boxes. If a user wanted an unusually large text box they could draw it at 'normal' size and then resize it, the initial recognition being retained.

### 4.03.4 Map Mode

When the user is satisfied with their sketch and wishes to transform it into the VB form designer they click the 'map' icon. This initiates the second stage of recognition (see 4.04.2 below). This act results in some strokes and words being joined and recognised as a single glyph; the default set of glyphs is shown in

Figure 28. The sketch is overlaid with labels showing the VB control types (Figure 25). The user may change the control type of a glyph by clicking on the label and selecting from the list of available controls. This list is built dynamically from the recognition engine's library. A glyph retains this new recognition. The user's alteration of a glyph's interpretation may affect other glyphs. Re-mapping the sketch resolves these ambiguities.

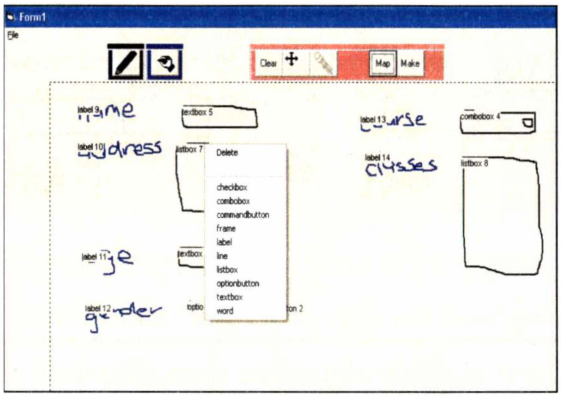



**Figure 25: Changing a Glyphs Recognition**

Once the sketch has been mapped the user may return to sketch mode or click the 'make' button to transform the sketch into a formal diagram in the VB form designer. The transformation process is explained in Section 4.05.

# 4.04 Recognition

There are two phases to sketch recognition in this prototype: stroke recognition and glyph recognition. Section 4.04.1 describes drawing stroke recognition and Section 4.04.2 describes how glyphs are recognised by the relationships between strokes. All recognition data are stored in simple text files that are read and written as required. Each user may have their own file(s) of stroke sets and rules. This prototype does not include character recognition.

## 4.04.1 Drawing Strokes

When the sketch-space is in draw mode, each pen stroke is parsed by the first part of the recognition engine as soon as the mouse-up event occurs. In order to achieve this aim we have implemented Rubine's algorithm (1991). However, the

recognition is hidden from the user as we did not want them to worry about recognition while they were in an early creative phase. Only the delete gesture initiates an immediate action.

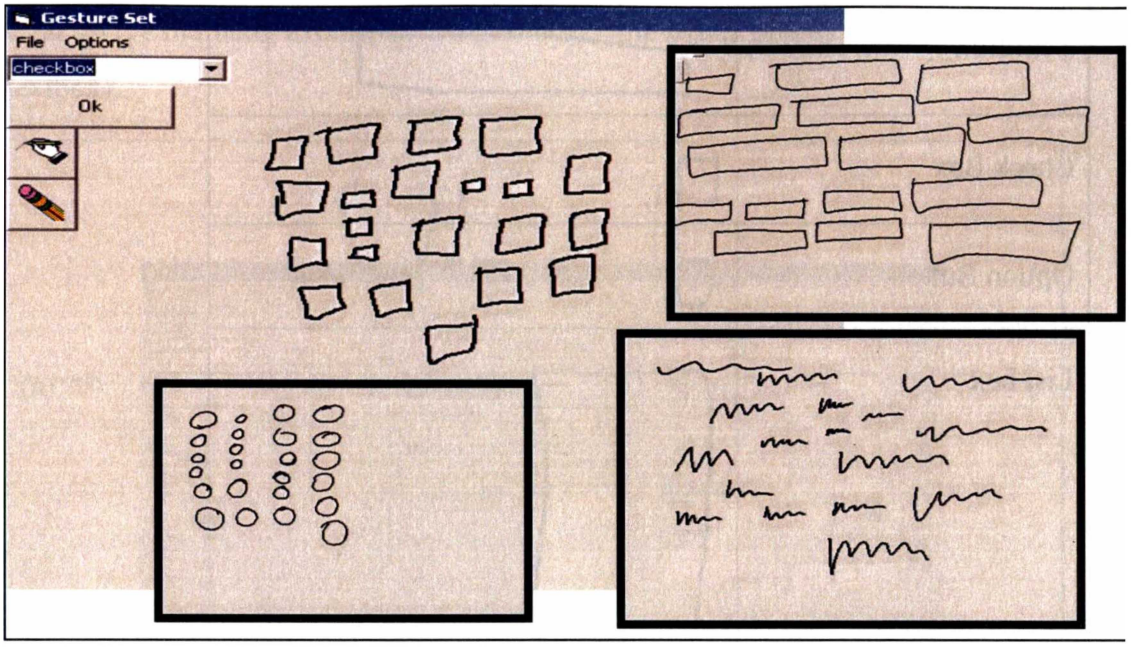



**Figure 26 Library Entries for Stroke Classes**

The system provides an interface allowing the user to enter training sets (Figure 26). From this interface the user can add a new class of strokes as well as add or delete example strokes from a class (a limited version of the sketch-space). There is also the facility to test the training sets to check that all the strokes in a class are classified as belonging to that class. If this test fails for some examples it indicates that there is likely to be some ambiguity problems when classifying this class of strokes.

The default system setup has six classes of strokes (Table 2). The text class is also the editing gesture for erase. This said, users can add any number of stroke classes, although it must be said that the system does not check for duplication or ambiguity. There are advantages and disadvantages of leaving this interface in an unrestricted state. On the positive side it allows users to specify the same shape name in relation to two (or more) distinctly different shapes. Conversely, the user could use two different names for the same or a very similar shape. They could

also define different shapes that the algorithm cannot disambiguate (Long et al., 2000).

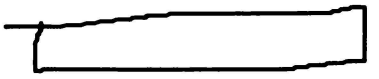
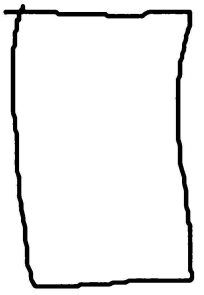
| Shape Name | Example |
|---|---|
| Text Box |  |
| Check Box |  |
| Option Button |  |
| List box |  |
| Line |  |
| Text |  |

**Table 2: Default Set of Basic Shapes**

Rubine's (1991) algorithm also provides two tests for rejecting a classification; a test for ambiguity and another for outliers. These are implemented; however the question then is; what to do when rejection occurs? As recognition is deliberately hidden from the user the first time that this information could be used is at the 'map' stage. It was felt, that at this point it is better to make some decision, than no decision. The recognition scores could be used to order the list of controls that the user can choose from, but a simple alphabetic list provides a more consistent interface. Therefore, the rejection tests are ignored in this prototype.

### 4.04.2 Relationships Between Strokes

A rule-based approach is implemented for establishing the relationships between strokes. Three types of relationships are of interest: joins, containers and singles.

A join exists when two strokes are joined to create a glyph, for example, a textbox containing text can in this way be defined as a button. Containers exists when the glyphs remain separate while one glyph 'contains' others. This is important in VB where frames are used to contain sets of radio buttons and the radio buttons in each set are mutually exclusive. Singles are single stroke glyphs (for example textbox).

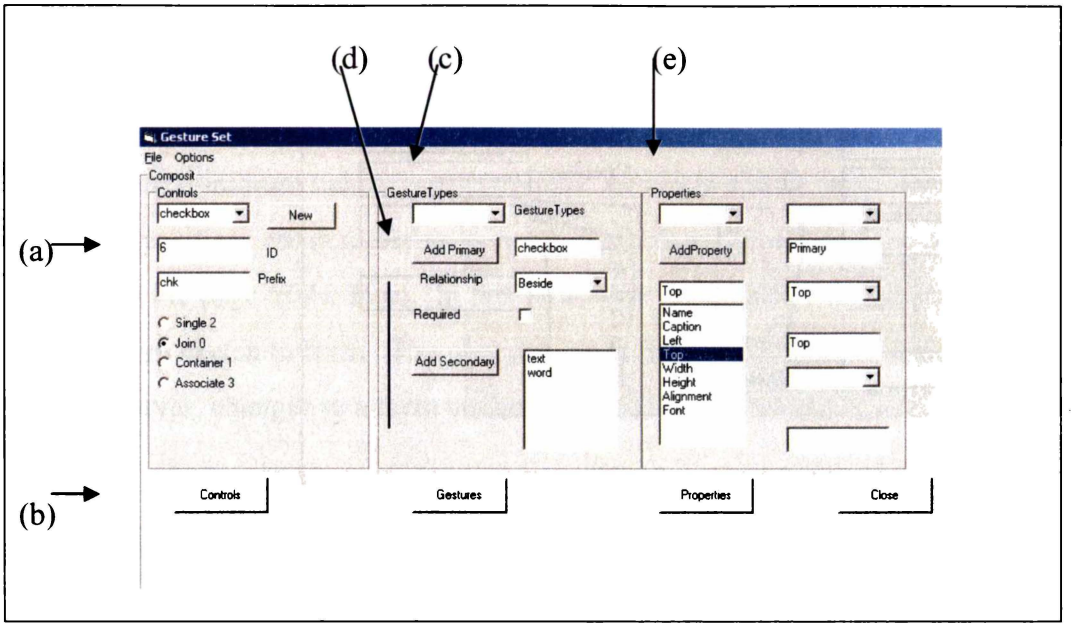



**Figure 27: Interface for Describing Glyphs**

The user is able to specify the rules for creating glyphs. First, the user enters the class of VB control (Figure 27a). Then they select the type of rule (Figure 27b): single, join or container. The next step is to specify the primary stroke (Figure 27c). If the control is a complex glyph the secondary stroke information is recorded (Figure 27d). The adjacency of secondary strokes is specified as beside, inside, left, right, above or below and maybe required or optional. When more than one secondary stroke type is specified they are automatically considered logical ORs. The screen shot above (Figure 27) describes a checkbox. A checkbox is a joined control that has a primary stroke of a checkbox (small square) that may have beside it a text stroke or a word.

There can be multiple rule definitions for a control type. Typically controls, such as a command button that requires a two-stroke join, has two rules, one specifying the basic stroke combination. 'A command button is a *textbox* that *must* contain a

*word* or *text'*. The other rule specifies the control type and is associated with optional secondary strokes. 'A command button is a *command button* that *may* contain *a word* or *text*'. Using the second rule the system can create the glyph if the user changes the control in map view.

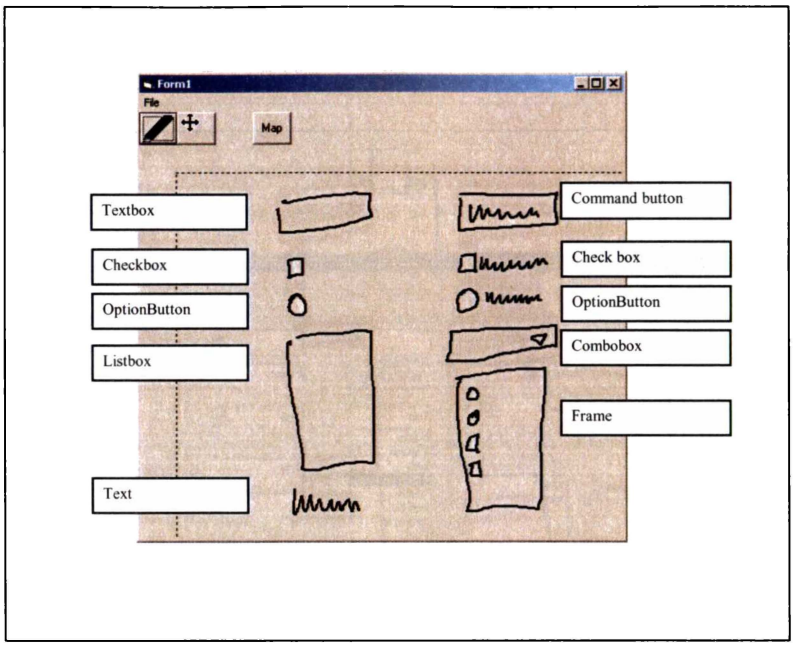



**Figure 28: Default Glyphs**

This collection of rules is applied when the user presses the map icon. The order of the application of rules is critical; the system searches first for containers, then joins and lastly singles. As with the stroke specification, there is no checking for contradictory rules. A default set of glyphs were provided for the usability testing (Figure 28).

# 4.05 Transformation

On the same interface as that which specifies glyphs (Figure 27e), the user can describe how the properties of a VB control are derived from the glyph. The system provides a list of the VB control properties and the stroke(s) attributes. The user can then specify the VB control property's value by specifying a fixed value or associating a stroke attribute with the control property. These connections can be constrained by setting minimum, maximum or divisor values.

The screen shot (Figure 14e) shows that the primary stroke's 'top' attribute becomes the checkbox 'top' property.

The VB form creation, *make*, can be executed once mapping has been completed by clicking the icon. The system first removes all controls from the target form and then checks through the sketch glyphs. Containers must be created and placed on the form before any controls fall within them.

The system creates a control and then applies the properties as specified in the rule-base. This is a simple process of assigning values, and checking for maximum or minimum values as appropriate. Controls within containers must have their positions recalculated as they are offset from the edge of the container rather than the edge of the form. In this prototype, the 'make' is a one way process from sketch to form. The user can move freely between the sketch and form, however, changes to a form are not reflected in the sketch.

## 4.06 Summary

This chapter has described the initial prototype that was both built to explore the technical issues of the software development and to be used for usability testing. The LIDS hardware provides a large digital whiteboard that accepts pen input. The sketch-space provides the basic interface for drawing and editing sketches, and also facilitates access to the map and make functions. Two stages of recognition are provided by the recognition engine: single stroke recognition and a rule-base to map strokes to VB controls. The transformation process takes the output from the rule-base to create the VB controls in the VB form designer.

This prototype achieved some important objectives. We were able to establish that the project was technically possible. The LIDS hardware provided satisfactory input from the technical perspective when the mouse events were used as the input. We successfully discriminated the types of shapes we wished to use with both Rubine's algorithm and a rule-base to combine strokes into meaningful glyphs. The system was integrated into the VB IDE as an add-in and can make VB controls, placing them on a form in the VB form designer.

Most importantly, the prototype gave us a vehicle to try out our ideas on target users. The next chapter describes the usability testing of this prototype and the subsequent enhancements planned for the next iteration of development.

# Chapter 5
# Usability Study of Initial Prototype

The usability study conducted on the prototype explained in Chapter 4 is described in this chapter. This usability study is a general assessment of the hardware and software. The main questions are: Do people find the LIDS hardware usable? How easy is it to use the sketch-space and is the resulting VB form likely to be useful? What other features are required? The usability study looked specifically at the central component of the system: the main sketch-space and resulting VB form. In the study, students were asked to design a form for membership details for a sports club. To do this they were required to complete the design process that included producing a low-fidelity sketch, recognition and the creation of a VB form.

The first section of this chapter describes the study design. This is followed by a description of the findings, which include a summary of the comments of the participants and the observations of the author. The chapter concludes with a summary of findings and an outline of the enhancements planned for the next prototype. Full copies of the participant guide and questionnaire data are contained in Appendix 1.

## 5.01 Usability Study Design

Nielson's (1994) 'discount usability study' methodology, which is widely used (for example, Knutilla et al., 2001), was adopted. This methodology has the advantages of simplicity while exposing most problems. A copy of the participant guide is provided in Appendix 1.1. The study participants had seen Freeform once, as a demonstration was given to the class the week prior to the study. The

six subjects each worked individually for about an hour. The subjects were members of an information systems degree introductory programming paper taught with VB6. The study was carried out in week ten of the course over a two day period. By this point of the course the students both had a reasonable understanding of the fundamentals of form design for a GUI environment and had used most of the intrinsic controls.

The study was conducted in the usability lab at The University of Waikato. The room was arranged with the LIDS in one corner and the computer on a table beside it. One video camera was placed beside the screen to record the user, while another was placed in front of the screen to record the screen and a third in the diagonally opposite corner of the room (Figure 29). These inputs were mixed with the video image to provide a four stream synchronous image (Figure 30).

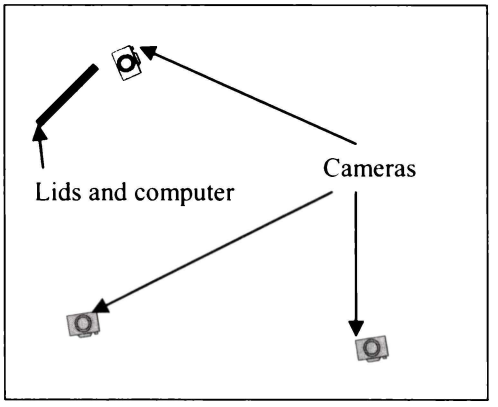



**Figure 29: Room Layout**

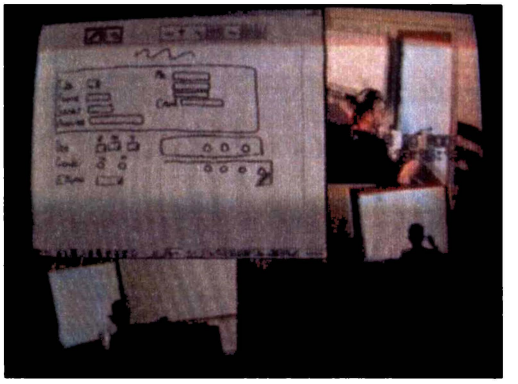


**Figure 30: Video Image Composition**

After each participant read and signed the ethical consent form they completed a short training session. The Freeform program was configured so that it first asked for the user's name. It then stepped through the shape library asking the user to add two examples to each set of shapes. This allowed the user to become familiar with the pen, the need for single stroke shapes, and their general shapes and sizes. The facilitator then explained how the recognition engine combined the basic strokes to make other glyphs (Figure 31).

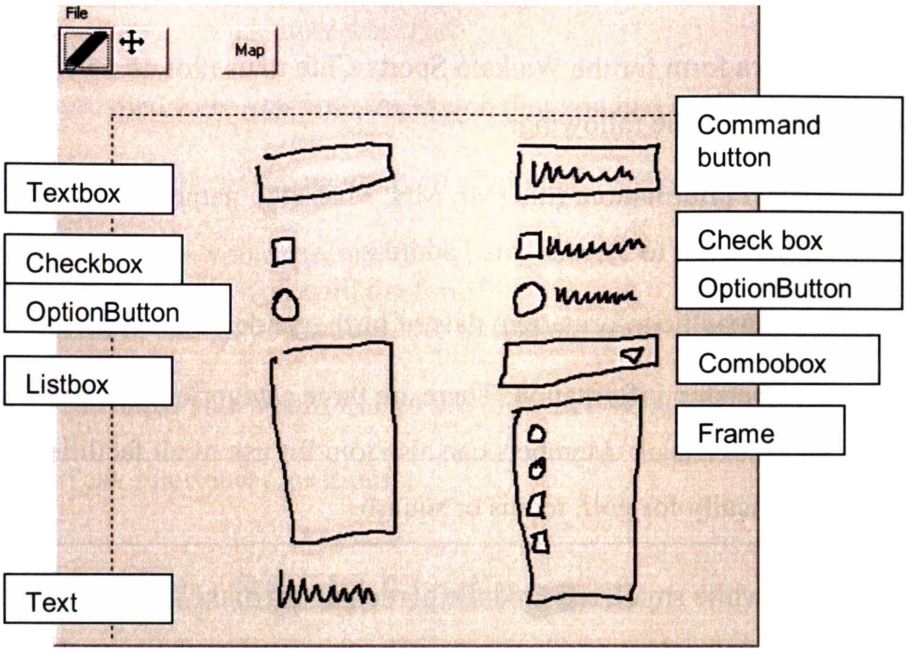



**Figure 31: Glyphs for Usability Study**

The training session also covered write mode, editing, mapping and making, with the user trying each of these functions. When the user felt comfortable with the basic interface he or she and facilitator together created a simple design. This practice sketch consisted of a heading, text boxes and labels for the student's name, id number and mark, and a command button for exit. Further instruction was given, as required, to the user during this process. The participant guide (Appendix 1.1) also included notes and diagrams on glyphs and editing. The training took about 15 minutes.

The participant was then presented with the following scenario for which they were required to create a form design. They were asked to 'think aloud'. The facilitator then moved away from the screen and sat down to observe, noting any difficulties that participants had and occasionally prompting particular participants to verbalise what they were doing or thinking.

**Design Scenario**

Please design a form for the Waikato Sports Club to use for member details. The form should show the following:

- contact information: title (Mr, Mrs, Miss etc), name, address, phone numbers (up to 3), and email address.
- Demographic information: date of birth, gender, ethnicity
- Membership information: There are three categories of membership, junior, senior or social. Members can also join for use of all facilities or specifically for golf, tennis or squash.

The scenario, while small, required the participant to make some choices about the general layout and types of controls. The requirement for up to three phone numbers offered an opportunity for three text boxes or lists. Date of birth likewise provided scope for variety (although the students had not learnt calendar controls). The demographic data (gender, ethnicity) and membership data (membership type and facility membership) could be recorded using a combination of radio buttons or check boxes, with frames being necessary if more than one set of radio buttons was to be included. The membership information deliberately provided an opportunity for more complex design decisions. When the participant had finished his/her design they were interviewed. The questions (Table 3) covered: basic drawing, basic writing, writing/drawing mode, editing, satisfaction with sketch, mapping and making. They were then asked about things they would like to change and enhancements they would like to see. Their comments are recorded in the next section.

**Post-Task Interview Questions**

**Basic drawing**

How did you find drawing on the lids screen (pen/ screen)?
Did you have any problems making the shapes you intended?
Was it difficult to create single stroke shapes?

**Basic writing**

How did you find writing on the lids screen?
Writing / Drawing Mode
Was it clear to you which mode you were in? Was it difficult to change modes? Did you have any difficulties with modes?

**Editing**

Did you use the scribble over to delete things? How was this?
Did you use the draw over to change things? How was this?
Did you move, resize, how was this?

**Satisfaction with sketch**

How satisfied were you with the sketch that you had created?

**Mapping**

Did you alter any of the recognition after mapping? How did you find this?

**Make**

Was the form as you expected it?
How satisfied were you with the form?

**If there were three things you would like to change what would they be?**

**What features would you would like to see in future versions?**

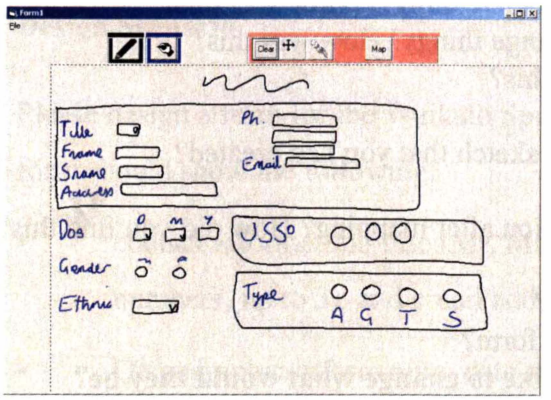
Table 3 Post-Task Interview Questions

## 5.02 Usability Study Findings

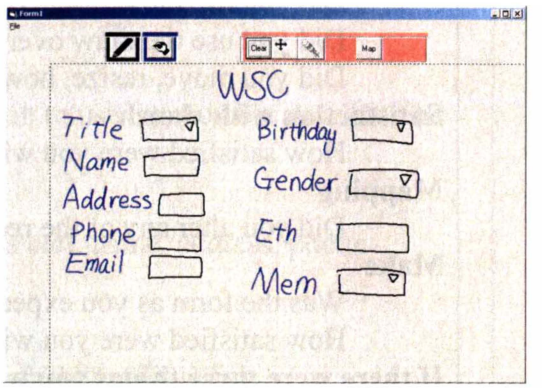
This section describes the usability study participants' responses to the questions and observations of the facilitator. All of the six people who participated in the study completed the task. Five did it with ease, one struggled (participant 4). His knowledge of VB controls seemed to be limited to text boxes and command buttons; it is noticeable that his design is simple and his responses to some of the questions (Appendix 1.2) show a lack of confidence. A screenshot of each participant's completed sketch is shown in Figure 32.

The participants said that they found the screen easy to draw on with two commenting that it was easier than a mouse as it was more direct and they did not find it hard to make the design look as they intended. We were concerned that single stroke glyphs might create a problem so we specifically asked about this. None of the participants said that they found this difficult, which our observations confirmed.

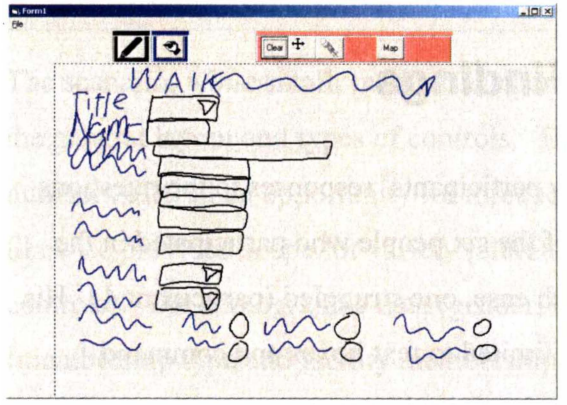
Writing was more difficult than drawing with two people commenting that the pen was too large and that the nib movement (about 5mm) was distracting. Others thought the ease of writing was irrelevant as there was no character recognition. Most of the participants said they were not worried about changing from the drawing to the writing mode, however our observation was that it did divert their attention from designing.



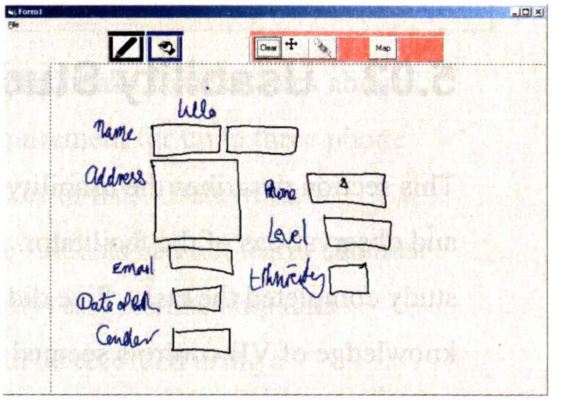

Participant 1



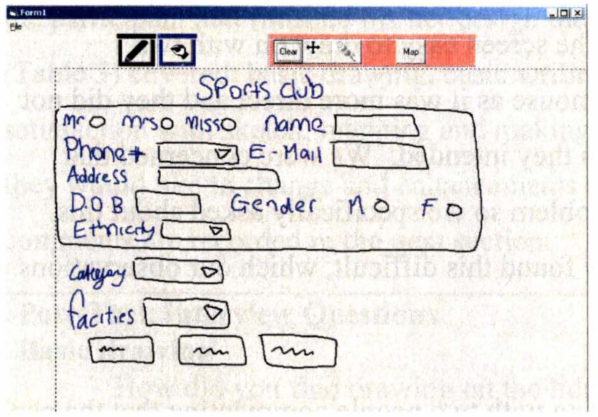

Participant 2



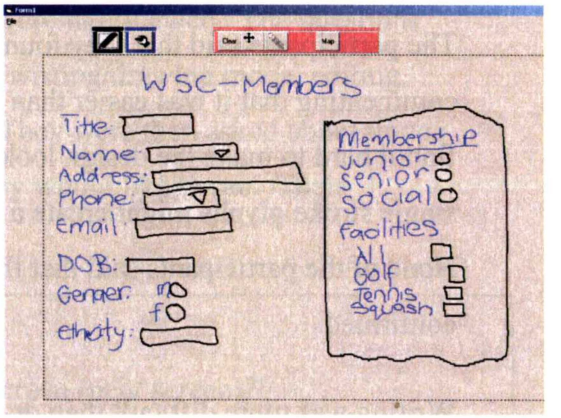

Participant 3




Participant 4



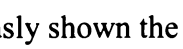
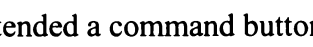

Participant 5




Participant 6

**Figure 32: Screenshots of Participants Designs**

If users forget to change modes they would then have to redo part of their design, so we prompted people if they were in the wrong mode. Most required only one prompt, but following this it was noticeable that they checked which mode they were in. We would like to eliminate this distraction. It is noticeable from the screenshots that participant 3 abandoned writing after a couple of words. During the interview, he commented that there wasn't any point as it was not recognised. Participant 5 made a similar comment.

The participants had been previously shown the erase gesture (  ). Two used it successfully, one didn't use it, the others had mixed results. For one, the program crashed. It didn't work for another because the bounding box did not cover the underlying strokes while for another participant the reverse condition happened when they had intended a command button (  ), but the greeking was interpreted as a delete gesture because it approximately covered the rectangle. Only one participant tried the draw-over function. It did not work for him, as the bounding box was not a close enough match. We found and fixed the program bug (the only crash during the study), but clearly this is an area that could do with further work.

The participants found deleting, moving and resizing satisfactory; during the first session, we noticed that participants (3 people) sometimes had trouble grabbing the handles. We made them a little bigger before the second session and this seemed to fix the problem. Four participants drew their sketches too large and needed to resize them. They felt that they needed to do this because of the pen size, the Mimio pen is larger than a normal whiteboard pen, and because they were working on a whiteboard.

The participants were then asked about both the sketch as an artefact and the generated VB form. They were asked if they would be happy to show their sketch to their lecturer or use it as a basis for creating their form. All except one expressed satisfaction. He (participant 4) said it would be ok if it was acceptable to others.

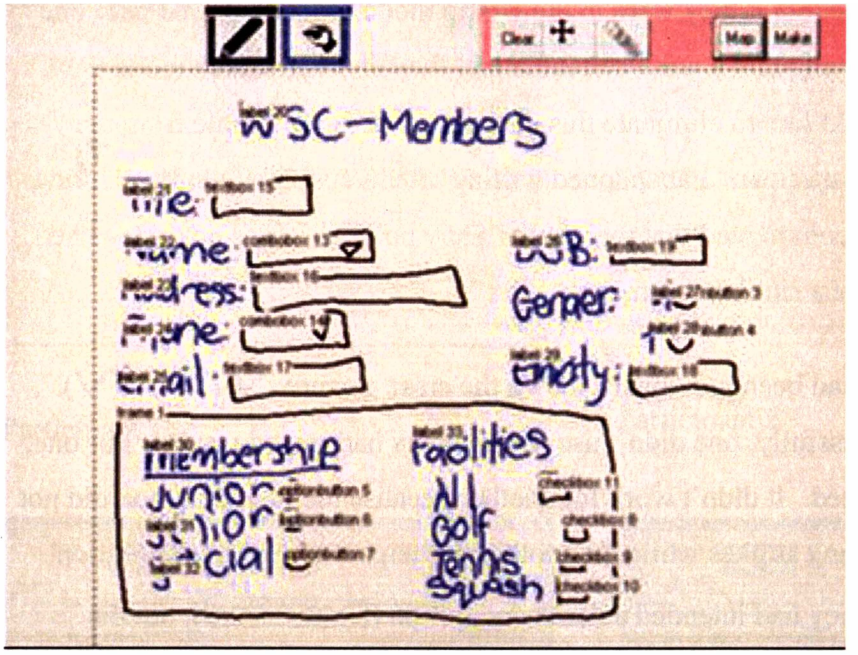



**Figure 33: Recognition Overlay of Sketch in Figure 32(6)**

Mapping is the test of the recognition (Figure 33). There was quite a wide range both of numbers of glyphs drawn and complexity of diagrams. The videotapes were reviewed to calculate the recognition rates. A summary of the number of glyphs and recognition rates is shown in Table 4. Overall, the group achieved an 86% success rate, with the best being 100% and the worst 62%. All were satisfied with the mapping and found correcting the recognition easy.

| Participant | Number of glyphs | Success percent | Number of corrections required |
|---|---|---|---|
| 1 | 21 | 62% | 8 |
| 2 | 9 | 100% | 0 |
| 3 | 16 | 87% | 2 |
| 4 | 9 | 100% | 0 |
| 5 | 16 | 94% | 1 |
| 6 | 16 | 94% | 1 |
| Total | 87 | 86% | 12 |

Table 4 Recognition rates

The VB forms that were created varied quite considerably with respect to how tidy they were. Where the sketch was tidy and the writing size consistent the form looked reasonable, however untidy sketches resulted in untidy forms. We asked about their satisfaction with the form; one participant said he would have liked it tidier and others included form tidying in their list of desired enhancements. We also felt that the form would be more satisfactory if widgets were aligned and standard font sizes were used for text. The form created from Figure 32(6) is shown in Figure 34 below.

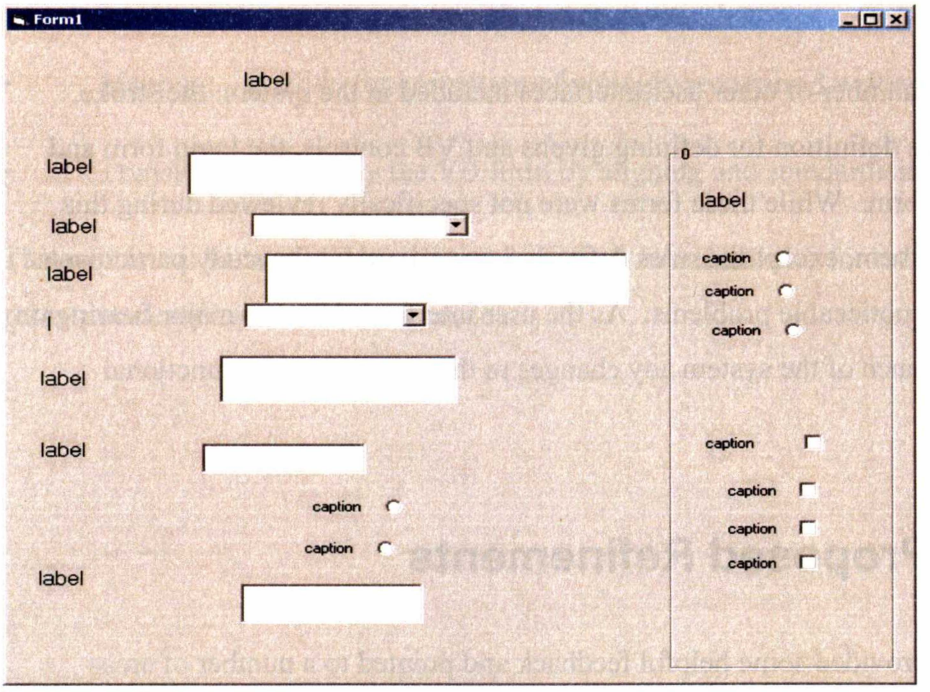



**Figure 34: Form from Sketch in Figure 32(6)**

The participants were asked to suggest three things they would like changed in the current interface. Three suggested a tidier VB form; the other most frequent request was for physical pen size to be reduced (2). One person wanted thinner virtual ink, another suggested D shaped handles for editing.

When asked what they would like to see in future versions, handwriting recognition was top of the list for five participants, while three suggested an undo feature would be useful. Other suggestions were: the ability to add words to a list or put default text into text boxes and a different glyph for the command button. Complete notes from the interviews are given in Appendix 1.2.

We also noted that two students clearly showed that they were considering the functional requirements of the program, even though they were aware that they would not be using these requirements. One (participant 6) had quite an extended 'self talk' about how she was going to design and code the membership categories, similar observations were made by Goldschmidt (1999) and Tversky (1999) When asked about this during the interview she said that she was considering how the underlying code would need to work.

## 5.03 Other Aspects of the Interface

There are a number of other user interfaces included in the system: the stroke library, rules definition for defining glyphs and VB controls, the login form and an options form. While these forms were not specifically reviewed during this study all of them, except the rules definition, were used by the study participants without any noticeable problems. As the user interfaces have no major bearing on the performance of the system any changes to them would be for functional reasons.

## 5.04 Proposed Refinements

This study provided some helpful feedback and pointed to a number of areas where the system could be improved. Most importantly the students were positive both about the idea and the system. In answering the main questions posed at the beginning of the chapter, we can say that the LIDS hardware is fine for drawing although some people find the pen a bit difficult to write with. In general the software worked well. The students found drawing and editing easy and fun. The recognition rates achieved for drawing were satisfactory with Rubine's (1991) algorithm providing reasonable shape recognition while the rule-base correctly inferred most relationships between strokes. The interface to correct recognition also worked well. The VB form generated from the sketches accurately represented what was drawn but did not meet users' expectations. It was also pleasing to note the existence of both 'self talk', and interaction between user interface design and coding requirements.

There are a number of areas where the system can be improved:

- Hardware – a better pen
- Drawing/writing – the need to change modes is distracting. Either eliminate modes or make it possible to change the ink mode by selecting and clicking an icon.
- Editing – provide undo and clipboard
- Recognition – recognise writing
- Mapping – provide for correction of wrongly recognised writing
- Transformation – tidy the VB form by aligning and standardising sizes.

These form the basis of the revision to basic tool functionalities for the second prototype described in the next chapter.

# Chapter 6
# Second Prototype

The prototype described in this chapter is based on the conceptual ideas of Chapter 3, and the knowledge gained from the first prototype and its usability testing. The goal of this iteration of the system's development is to develop a tool that we can use to evaluate the utility of computer supported low-fidelity sketching.

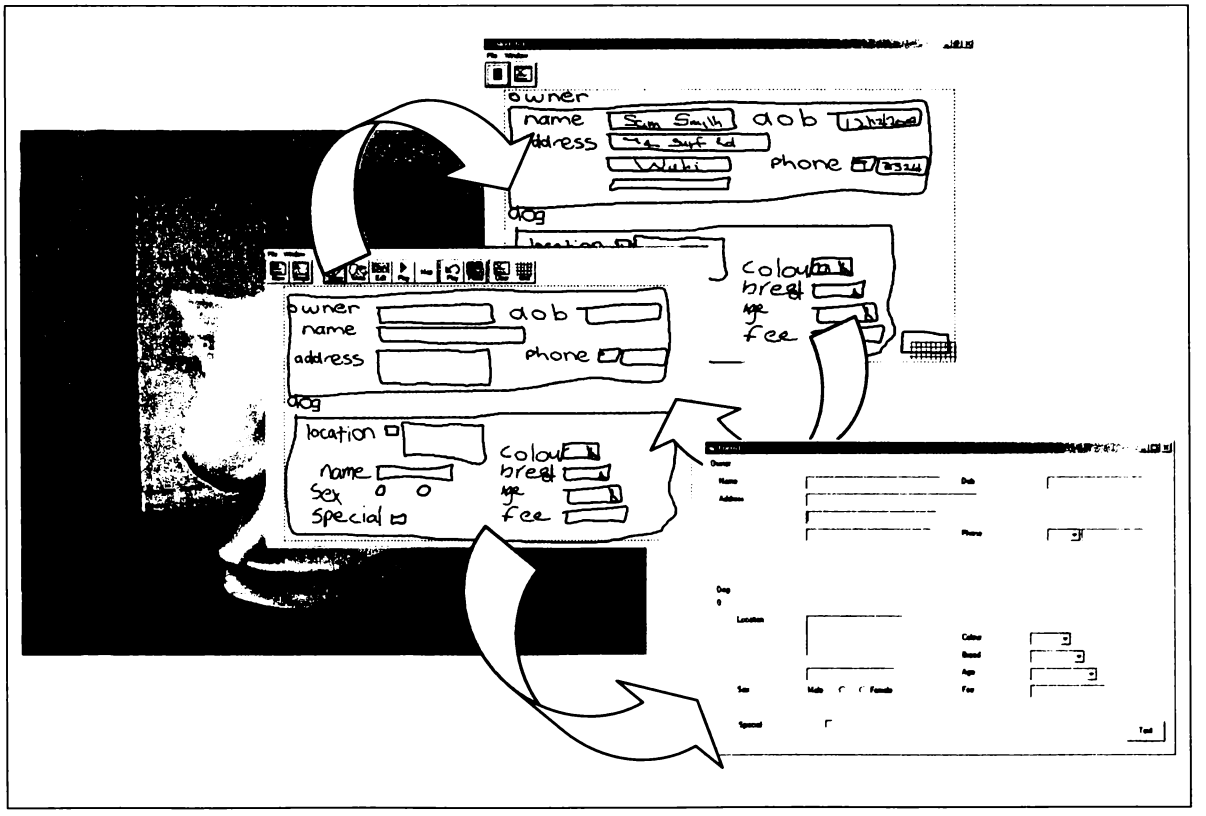



**Figure 35: Design Support Using the Second Prototype**

Chapter 3 listed five important concepts for a low-fidelity design tool. The first of these was a low-fidelity, unconstrained drawing space where users pen directly onto the display surface. The first prototype imposed some restrictions in order for the system to be able to recognise ink. We were concerned about the affect of

single stroke drawing gestures, but this proved not to be a problem. We were also concerned about the need for drawing and writing modes; this is a problem that this prototype needs to address.

The second concept identified was a group space. The LIDS setup worked well during the usability study except for the pen, which was too large and had too much movement in the nib. Also, as described in Chapter 3, buttons on the pen that can be used to change modes (edit, erase) would be useful. A survey of available hardware at the beginning of 2002 found there had been few advances in the functionality of digital whiteboards so the LIDS setup remains unchanged.

The third concept identified was support for editing and storing sketches. The usability testing of the first prototype provided information on what users expect for editing in a low-fidelity environment. The recommendations from the usability test are incorporated in this prototype.

The fourth concept identified involved a way for designers to interact with the design while the design is still rendered as a sketch. While the first prototype provided for a single sketch, this prototype supports multiple sketches, as well as implementing both the storyboard and run mode described in Chapter 3.

Finally, there is the concept of integrating the software into an IDE so that users can progress from informal to formal design with a minimum of effort. We achieved the integration into VB6, finding that the transformation from sketch-to-VB form is very quick. However, users suggested, and we agree, that the formal design would be better if it was tidier. Also the users reinforced the need for character recognition.

The designed process steps that Freeform explicitly supports has been expanded to include interactive checking (Figure 35). This chapter highlights the changes that have been made since the development of the initial prototype.

# 6.01 Sketch-Space and Storyboard

The sketch-space of the first prototype has been enhanced and the system supports multiple sketches; the new storyboard provides an overview of all sketches and allows the creation of navigation links between sketches. Users can move freely between the sketches and storyboard by clicking the appropriate icon (Figure 36).

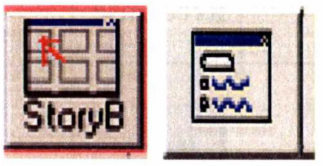



**Figure 36: Storyboard and Sketch Icons**

## 6.01.1 Sketch-Space

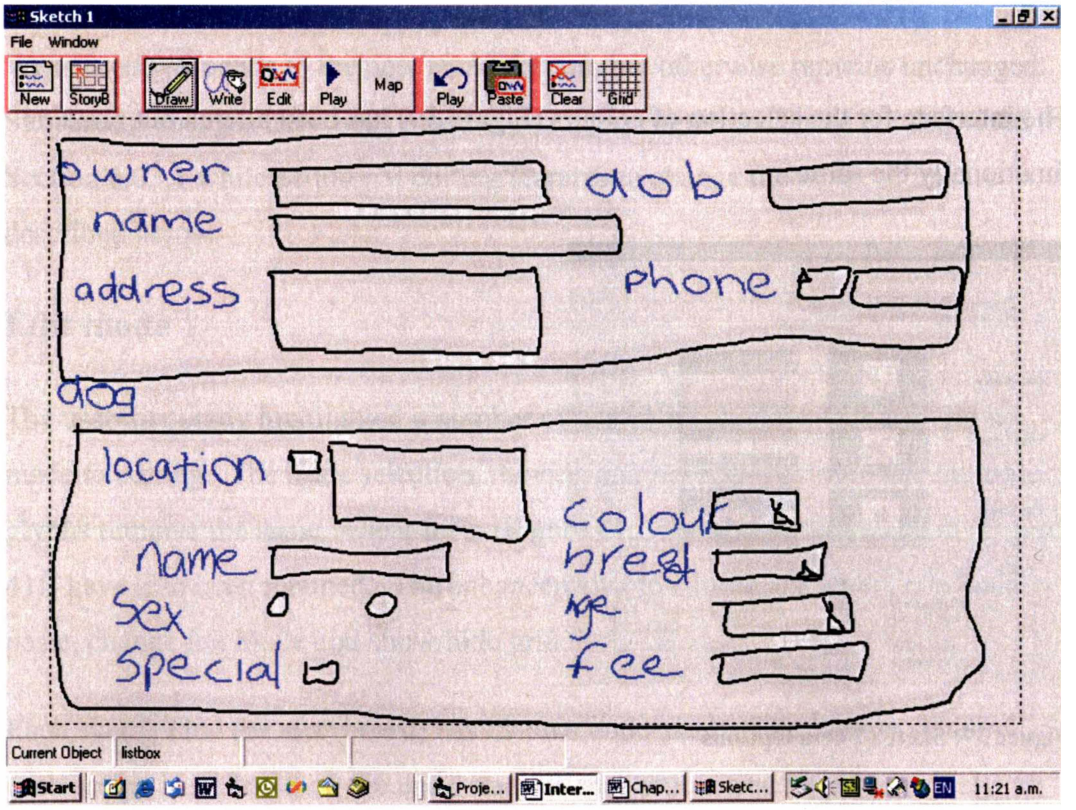



**Figure 37: Form Sketch-Space**

The sketch-space (Figure 37) retains the draw, write, edit and map modes of the previous prototype. A run mode has been added and is described in Section 6.02.

The icons for these modes are shown in Figure 38. Each of these modes are visually different and are identified according to ink colour, pen icon and/or look of the sketch, depending on the mode. The appearance of draw, write, edit and map are the same as the previous prototype. The cursor for run/play is shown in Figure 38 and the ink colour can be selected from the options interface described below.

| | Draw | Write | Edit | Play | Recognise |
|---|---|---|---|---|---|
| Icon |  |  |  |  |  |
| Cursor |  |  |  |  |  |

**Figure 38: Mode Icons and Cursors**

The interface for the selection of colours (Figure 39) has been altered but remains functionally the same.

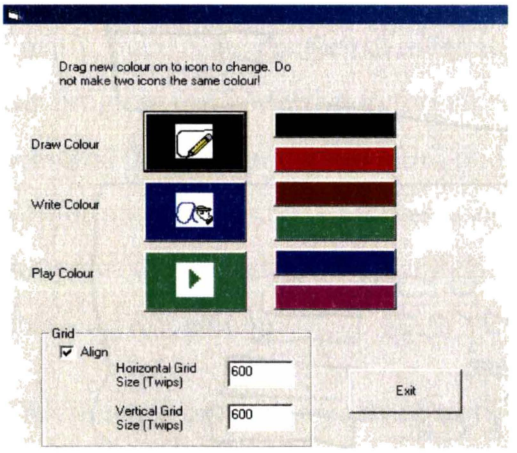



**Figure 39: Sketch Form Options**

The sections below describe the changes to each mode.

## Draw mode

The overloading of the erase and text gestures in the first prototype caused difficultly with recognition and usability. To correct this, the erase gesture is now a unique gesture (Figure 40). This change has the added benefit of making it user configurable, meaning a user can replace the sample gestures in the library with something quite different.

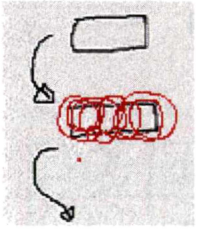


**Figure 40: Delete Gesture**

## Write mode

Write mode responds to the new erase gesture, but otherwise remains unchanged. Substantial changes have been made to writing recognition. These are described in Section 6.03.1 while additional editing features to change the mode of ink are described below.

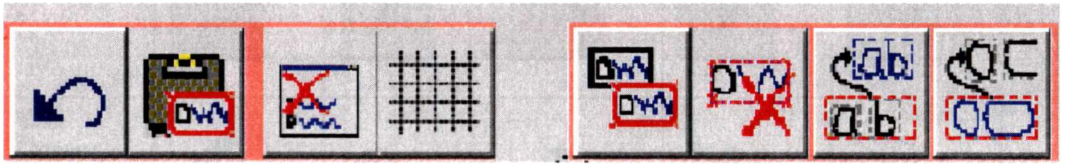
## Edit mode

The usability study highlighted a number of useful enhancements that could be made to editing. The basic selection, moving and resizing of single and multiple glyphs remains the same. Clear form (Figure 41c) and the delete selection (Figure 41f) have also been retained. The enhancements to editing are: undo, copy and paste, change ink mode and show/hide grid.

Each sketch (and the storyboard) has its own undo stack. The 'before state' of each action is recorded on the undo stack so that actions can be progressively reversed by clicking the undo icon (Figure 41a). The user can copy selected ink onto the clipboard by clicking the icon (Figure 41e) and then paste to the same or a different sketch within the project (icon Figure 41b).

As we have been unable to resolve the drawing/writing modality this prototype incorporates the ability to change the mode of ink. The icon Figure 41g changes selected ink into a word (letter spacing was not examined) while Figure 41h changes selected ink into individual glyphs. Ink is re-recognised after a mode change.

A grid can be displayed on the drawing space and is toggled on/off by clicking the icon Figure 41d. The grid size can be changed by the user (Figure 39). The grid is used to assist with form tidying, which is described in Section 6.04.



(a) (b) (c) (d) (e) (f) (g) (h)

**Figure 41: Editing Icons**

## Map mode

Glyphs in map mode are snapped onto the grid (Figure 42) as described in Section 6.04. Because of this, the grid is automatically turned on when a sketch is displayed after mapping. This version includes word recognition (see 6.03.1). In map mode, the user can alter the word by selecting *change caption* from the dropdown menu (Figure 42).

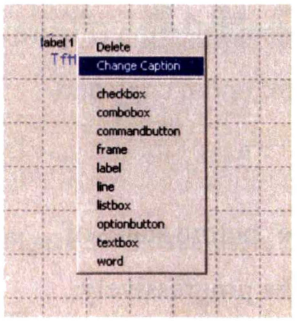



**Figure 42: Glyph Menu**

This brings up the vocabulary list and an on-screen keyboard that allows the user to choose or add a word (Figure 43).

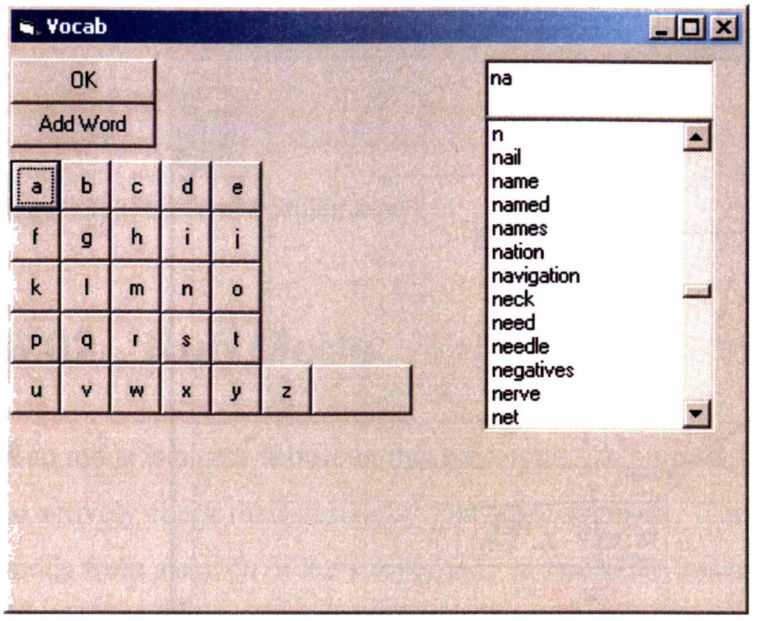



**Figure 43 Vocabulary form**

As with shape recognition, once the user has overridden the automatic recognition the system remembers the user-assigned word. However, this facility is lost if the user changes the ink type.

### 6.01.2 Storyboard

The storyboard provides an overview of all the sketches in a project; each sketch is shown in miniature (Figure 44). By default the sketches are displayed in the order they were created with empty slots at each end of the storyboard. Users can reorganise the storyboard by dragging a sketch to an empty box and the sketch's new storyboard position is permanently retained. It is also possible to delete sketches by dragging them to the trashcan.

The user can establish run-time navigation links between sketches by dragging a line from one sketch to another. The start point of the line becomes the navigation hotspot and the finish point is the destination sketch. The destination is shown as a half circle at the edge of the sketch, as a link to a sketch, rather than a specific point on the sketch. The link can be moved or trashed by dragging an end point.

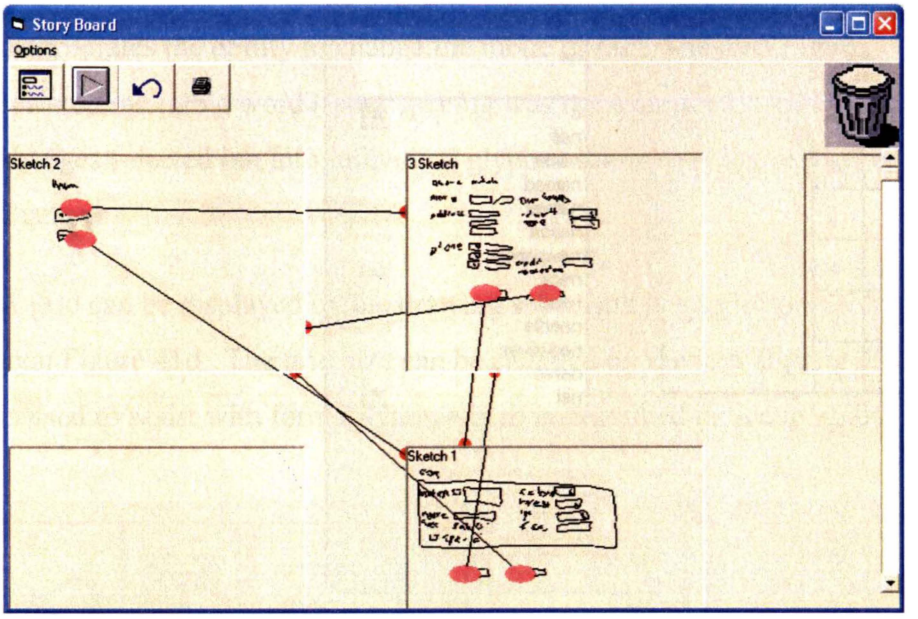



**Figure 44: Storyboard**

The possible interactions on the storyboard are limited, therefore the software can disambiguate the intentions of the user. As the pen moves around the interface it indicates the type of action that results from a pen-up at that particular location by changing the cursor (Figure 45). If the pen-down is in a sketch (and not on a link source) the cursor is set to auto move. If the pen then moves to another sketch the cursor changes to link, or if it moves over the trashcan the cursor changes to form delete. If the pen moves to a blank storyboard slot the cursor changes to form move. If the pen-down is on a link source or destination spot the cursor starts as a link move, changing to link delete if the pen moves to the trashcan.

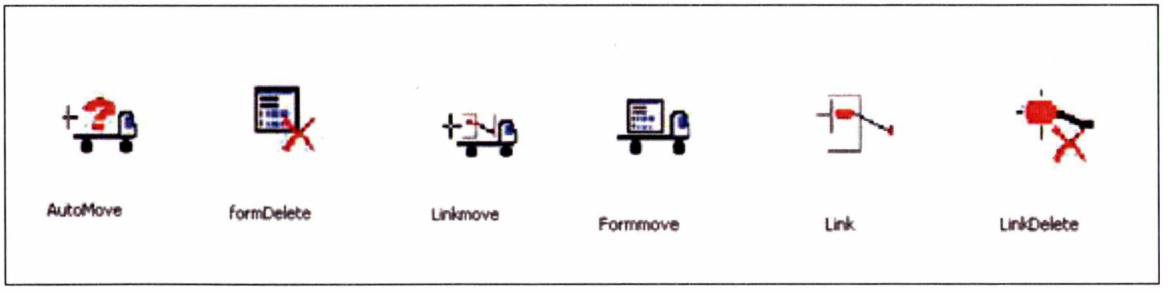



**Figure 45: Storyboard Cursors**

The user can specifically choose the type of action they intend by turning auto-mode off and selecting from the icons (Figure 46). From left to right these icons are: create link, move link, delete link, move form and delete form.

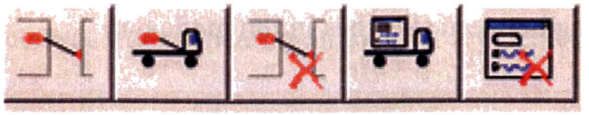


**Figure 46 :Storyboard editing icons**

## 6.02 Run Mode

Run mode is a new feature in this prototype. Its purpose is to encourage students to actively check their design by 'playing computer'. The user can enter run mode from a sketch or the storyboard. In run mode it is as if a clear overlay is placed over the sketch. The underlying sketch is inert; hotspots are placed on the overlay at source points for the inter-sketch links that have been established in the storyboard view. The user can draw or write on the overlay and navigate between forms by tapping the hotspots (Figure 47).

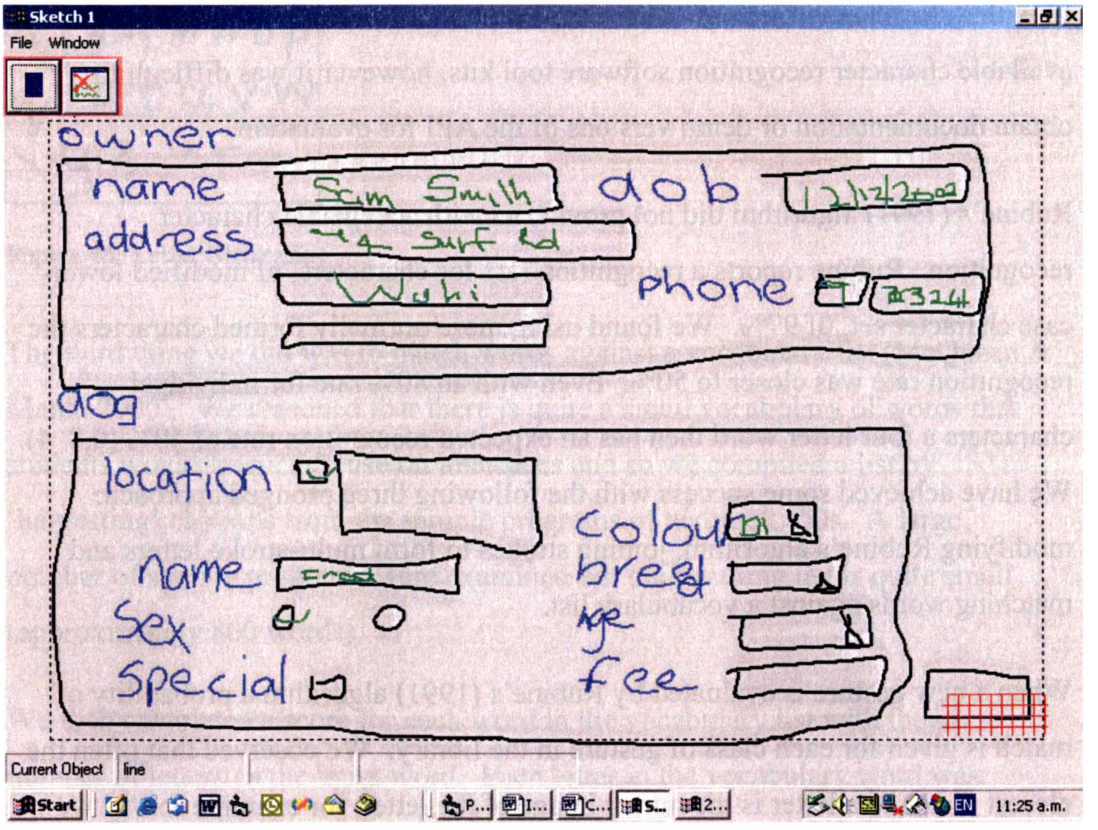



**Figure 47: Run Mode View of Figure 37**

Inking in run mode is passive (no recognition) may be cleared by clicking the clear icon. Ink is retained while moving between sketches in run mode but cleared when the user exits run mode.

## 6.03 Recognition

There are three parts to recognition in this prototype: stroke, word and combining glyphs. The recognition of shapes is substantially the same as that of the previous version. Providing word recognition was a major goal of this prototype; how this was achieved is described below.

### 6.03.1 Word Recognition

We decided not to use proprietary character recognition. We found that the character recognition in Microsoft Windows Text-Services Framework™ did not work with the Mimio mouse driver and that the programming API was not a good fit with a sketch environment. There are a number of other commercially available character recognition software tool kits, however it was difficult to obtain documentation or demo versions of the API for evaluation.

Rubine's (1991) algorithm did not provide us with acceptable character recognition. Rubine reports a recognition rate for characters, of modified lower-case character set, of 97%. We found using more normally formed characters the recognition rate was closer to 50%. Even with an 80% rate for individual characters a four letter word then has an expected recognition rate of 40% (0.8^4). We have achieved some success with the following three pronged approach: modifying Rubine's algorithm, joining strokes to form multi-stroke letters and matching words against a vocabulary list.

When a new gesture is evaluated by Rubine's (1991) algorithm a probability of match is given for each class of gesture in the library. We observed that often the closest match to a letter is the mirror image of the letter, for example for 'b' we would get 'd' or for 'm' we would get 'w'. We added two features to the feature set; the x and y points that represent the point-of-balance of the ink. This gave an

improved rate for the best matched letter and also improved the likelihood of the correct letter appearing in the first few letters.

The second action we took was to allow two strokes for some letters. The system seeks to recognise the lower-case letters a-z. Most letters are expected to be formed by a single pen stroke, however for the letters 'i' and 'j' if a dot is present it is combined with the stroke beneath it and the appropriate letter added to the recognition list before the basic stroke letter. For example, an 'i' may be formed by a 'l', which is recognised as an 'l' and a '.'; 'i' is then placed in the recognition list before 'l'. The letters 'f', 't' and 'x' may be recognised by combining the two intersecting strokes, although 'f' and 't' also have a single stroke form. A complete set of letter strokes used in the studies is shown in Figure 48. When the stroke recognition and combinations are complete each letter is represented as a list of possible matches in descending order of likelihood.

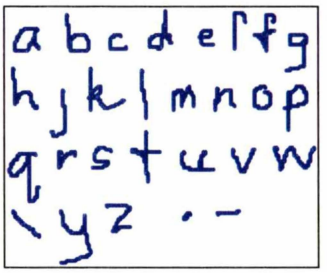


**Figure 48: Letter Stroke Set**

The third thing we did was to match words against a vocabulary list (McQueen & Mann, 2000). We reasoned that there is quite a small vocabulary of words that students would frequently use on interfaces and so we compiled a list by 'harvesting' captions from the sample programs of textbook CDs. A large number of sample programs were examined but the resulting list is quite small (approximately 800 words).

We then calculated a score for each word in the vocabulary list with the same number of letters as the input word. Each letter in the vocabulary word was assigned a score equal to its position in the list of the input word's letter list produced by Rubine's (1991) algorithm. The word score is the sum of the individual letter scores. The lowest scoring word is the match. If no word scores

an average letter placement of less than third place then the word is deemed not to have been matched. An example of this process is shown in Figure 49.

Example

| Input | Letter list from algorithm |
|---|---|
| t | t,q,b,y,h,d,p,n,c,k,f,x,a,g,j,e,f,m,u,z,l,i,v,s,o,w,r,-,.,text,delete |
| h | h,k,b,y,t,n,f,g,q,x,j,m,f,u,l,i,c,z,a,p,d,e,s,w,v,r,o,text,-,.,delete |
| e | t,n,f,m,w,z,e,r,x,u,d,v,-,y,q,p,a,h,k,c,b,f,.,g,text,s,o,j,l,i,delete, |

Scored against

"the" = 1 + 1 + 7 = 9

"add" = 13 +19 + 11 = 43

"age" = 12 + 8 + 7 = 27

"the" is the lowest scoring match, it is within the average of third place (even though the 'e' is in place 7) so the word is matched

**Figure 49 Example of Word Matching**

# 6.04 Transformation

The VB forms generated by the first prototype were rather untidy. Glyphs that appeared to be the same size and aligned on the sketch turned out to have significantly different sizes and were unaligned on the form. Three enhancements were made to improve the neatness of forms.

First, two extra variables were added to the information held for the generation of control properties; one held the type of rule to apply and the other a value for that rule. A property can now have an associated rule to set a fixed, minimum, maximum, or unit value. This rule and its associated value are used for height and width properties. The unit value is particularly useful for setting heights of controls. This property is often derived from the bounding box, which is generally larger than the intended widget size. The units used are the integer quotient (not rounded) of the bounding box. For example, a textbox may have a unit height of 500 twips (150 twips ≈ 1 pixel). Any textbox less than 999 twips (500 x 2 -1) is set to 500 high. Those between 1000 and 1499 are set to1000 high.

Second, a grid is set onto the form and all gestures are snapped to the grid. The grid size can be set by the user (Figure 39); generally a grid of 400 to 600 twips is appropriate. The snap-to-grid favours taking a glyph to the top and left of its current position (60:40). The repositioning of glyphs onto the grid is done at the same time as the second stage of recognition (map). It is then a simple task for a user to move any gestures that have not been aligned correctly.

Thirdly, attempts at intelligently interpreting font size were abandoned as they gave poor results. All text is created at the default size. The resulting form is uniform with controls vertically and horizontally aligned, and which are of standard sizes (Figure 50).

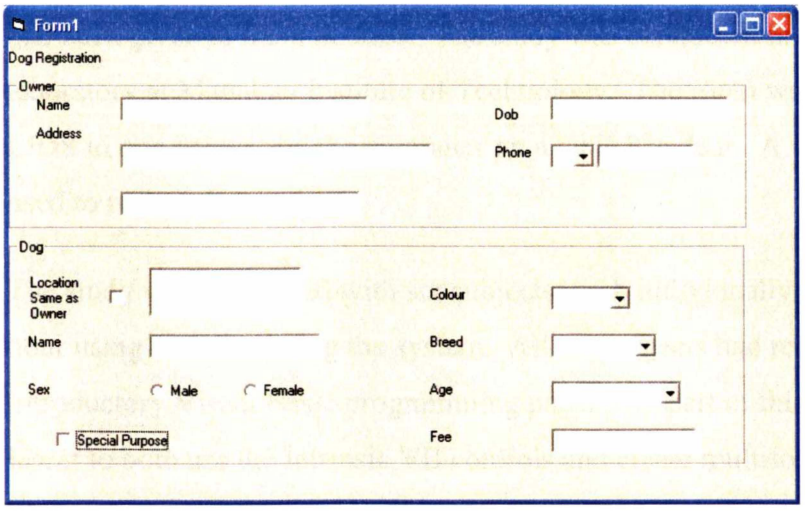



**Figure 50: Generated Form from Figure 37**

## 6.05 Summary

This second prototype uses the same hardware setup as the first prototype but provides more functionality and features. The sketch-space has had a major overhaul and now supports both multiple sketches and a storyboard. The editing on a sketch has also been enhanced providing undo, copy and paste, and ink mode changes. The storyboard shows thumbnails of each sketch. The sketches can be moved around the storyboard and navigation links established between sketches.

The run mode added to the system is to encourage students to interactively check their sketches using scenarios. The sketch to form transformation process is greatly improved by aligning controls onto a grid and standardising the sizes of controls. Also basic word recognition is added to the system as this was an overwhelming request from users.

This prototype addressed the critical issues uncovered in the first usability test and now provides most of the features described in Chapter 3. The next chapter describes the usability study we undertook to check if there were any major usability issues that need to be addressed prior to the evaluation study.

# Chapter 7
# Second Prototype Usability Study

The second prototype includes: a number of changes to the main sketch-space, two new interfaces, a storyboard for viewing multiple sketches and a run mode for checking designs. This chapter describes the usability study that was undertaken both to check whether these changes had been beneficial and that there were no critical usability problems that needed to be addressed prior to the evaluation study. Of particular interest was whether (1) the new editing functionality (undo, copy and paste) worked as intended and (2) the word recognition was sufficient to encourage the students to write.

## 7.01 Usability Study Design

This study is very similar to the study described in Chapter 5, which used the Nielsen's (1994) 'discount usability study' methodology. All of the study participants had seen but not used the interface once before, as a demonstration had been given to them in class. The study was conducted in a research laboratory at Manukau Institute of Technology. The room was arranged with the LIDS in one corner and the computer on a table beside it. A video camera was used to record the sessions.

The study was conducted with six subjects, each individually spending about an hour using and discussing the system. All the subjects had recently completed an introductory Visual Basic programming paper. As part of this course they had learnt to both use the intrinsic VB controls and create multiform projects. A copy of the participant guide is provided in Appendix 2.

After a participant read and signed the ethical consent form he/she were given preparatory training. This training covered the draw/write mode, editing, creating multiple sketches, the storyboard and links, the run mode, mapping and making. The participant then had the opportunity to try out each of these functions. Then the participant and trainer together created a simple two-sketch design. The first sketch consisted of a heading, text boxes for the student's name, id number and mark, and buttons for navigation to the other form and exit, the second sketch was a 'course form'. On the storyboard navigation links were created between the two forms and then the Freeform run mode was used to check the design. Further instruction was given to the participant, as required, during this process.

The participant was then presented with the following scenario for which he/she was required to create a design.

> Please design two forms for the Manukau Sports Club. The first form is a menu to take the users to forms for member details, member accounts, asset management, and facilities bookings. The second form is the member details form (you will not do the others forms due to time constraints), it should show the following:
>
> - Contact information: title (Mr, Mrs, Miss etc), name, address, phone numbers (up to 3), and email address.
> - Demographic information: date of birth, gender, ethnicity
> - Buttons to return to the main menu and to go to the member accounts form

This problem is based on the one used in the previous study, however in this instance, the member detail form requirements have been simplified so that another form could be included in the problem without extending the time required for a session. The first form is a simple menu (Figure 51). The second (Figure 52) is the members' details form used in the previous study. After completing the sketches, the participant used the storyboard to draw links between the sketches (Figure 53), and then used the run mode and a scenario that we provided (Appendix 2) to check their design.

The facilitator observed the sessions, taking notes on the participant's actions. After a participant had completed the design task, they were interviewed. The questionnaire covered basic drawing, basic writing, the writing/drawing mode, editing, the storyboard and run mode, satisfaction with the sketch, and mapping and making (Table 5). The participants were then asked both about things they would like to see changed and what enhancements they would like to see added to the Freeform tool. Their comments are recorded in the next section.

**Post-Task Interview Notes**
We would like to understand how you found the experience of using the sketch interface
**Basic drawing**
How did you find drawing on the lids screen?
Did you have any problems making the shapes you intended?
Was it difficult to create single stroke shapes?
**Basic writing**
How did you find writing on the lids screen?
**Writing / Drawing Mode**
Was it clear to you which mode you were in?
Was it difficult to change modes?
Did you have any difficulties with the modes?
**Editing**
Did you use the scribble-over to delete things? How was this?
Did you use the draw-over to change things? How was this?
Did you move, resize? How was this?
**Satisfaction with sketch**
How satisfied were you with the sketch that you had created?
**Run mode**
Did you find run mode useful?
Did you decide to change anything on your sketch after doing a 'run through?'
**Mapping**
Did you alter any of the recognition after mapping? How did you find this?
**Make**
Was the form as you expected it?
How satisfied were you with the form?
If there were three things you would like to change, what would they be?
What features would you like to see on future versions of Freeform?

**Table 5: Post-Task Interview Questions**

# 7.02 Usability Study Findings

This section reports both the participants' responses to the questionnaire and the observations of the author. Five of the six people who participated in the study completed the task; the sixth worked more slowly and ran out of time, so only one of his sketches was converted to a VB form.

As with the previous study, the participants found it easy both to draw on the screen and to make the single stroke glyphs. Figure 51 and Figure 52 show one student's sketches of the menu and members' details forms. Screenshots for all participants' designs are included in Appendix 2.

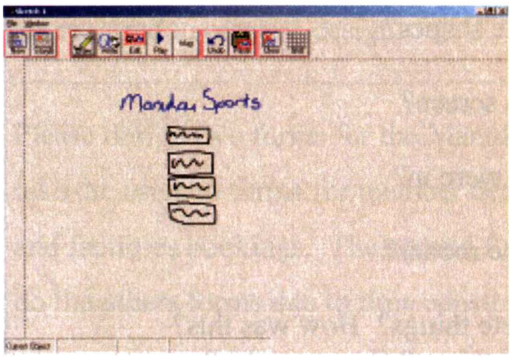



**Figure 51: Menu**

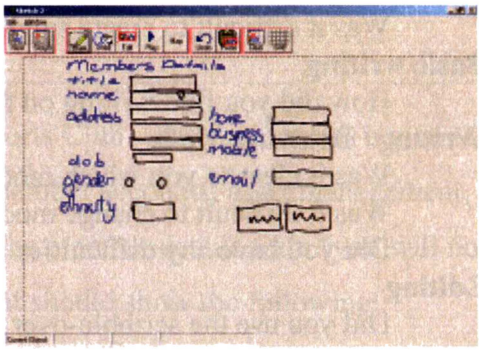



**Figure 52: Members Details**

Two of the participants commented that writing was difficult, given that writing 'correctly' is important if one is expecting the software to recognise what has been written. The software is trained to recognise single stroke lower-case letters. This was not a problem for most letters, however there are alternative ways to form a number of letters. For example, 'k' can be formed with one stroke '' or two '  '. We also observed that it is more natural to capitalise the first letter of a label, for example 'Address' is more likely to be written than 'address'. With the system's limited character recognition we could have chosen to either spend more time training the users or we could have left them to write as they wished, thus accepting that the recognition would be poor. We chose the second approach. The algorithm could not recognise much of the writing because some participants used capital letters or joined two letters together and this went beyond the program's scope; word recognition rates were very low. However, it is easy to change the

recognition of a word. Although only one student suggested better word recognition as being one of the things they would like to see improved, we see this problem as an outstanding challenge.

The software still has separate writing and drawing modes that concerned us after the first study. However, swapping modes seems to be less of a problem tha expected and elicited fewer comments than in the previous study. Furthermore, the software now has the facility to change the mode of ink. We postulate that word recognition has made the modes more valued and therefore more memorable.

The new erase gesture has overcome the problems experienced with the overloaded 'text'/'erase' gesture; two participants commenting that this was 'cool' and 'exciting'. Two participants chose to use undo instead of erase. Consistent with the previous study the 'draw over' was rarely used. Resizing and moving was a little slow because of the machine used for the study, but otherwise satisfactory. Undo was well received and frequently used.

The sample problem used in the study required navigation links to be created on the storyboard (Figure 53). Participants commented that the storyboard is very simple to use. The participants checked their design in run mode by pretending to register a new member. They had no difficulties with this.

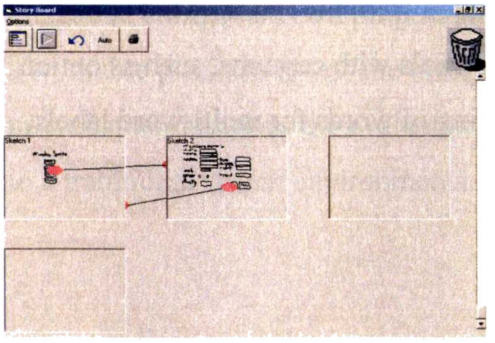


**Figure 53: Storyboard**

When mapping the glyph, the recognition rate remained similar to the previous study. The participants found the correction of the miss-recognised glyphs and words easy to accomplish. Mapping also aligns glyphs to a grid and most glyphs

were aligned correctly. The participants had no problem moving incorrectly aligned glyphs to an appropriate position.

The forms resulting from this prototype (Figure 54) were much tidier than the previous version as the controls were now of standard sizes and aligned. We did not receive the suggestions as were offered in the first study that we needed to produce a tidier VB form. In fact, one student who had been dissatisfied with his sketches, because they were too untidy, was pleased with the VB forms that had been created from his sketches.

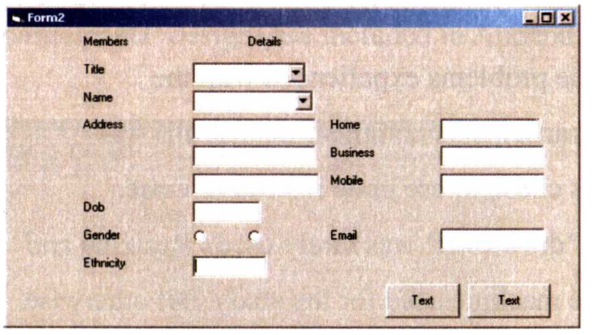



**Figure 54: Generated Form from Figure 52 Sketch**

Our request for suggestions for changes and enhancements brought a wide variety of responses. The physical pen is difficult to write with. This was commented on by two participants. The only other suggestion made by more than one person was that the function of the button icons needed to be clarified. Some of the suggestions involved the need for: code generation from the run time hot spots, the naming controls while doing the sketch (controls with captions, such as option buttons are named automatically), and the joining of words for multi-word labels.

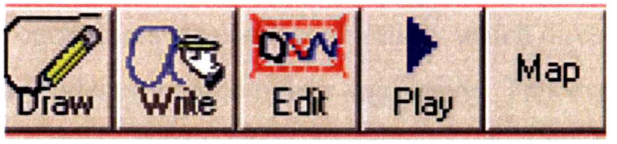
## 7.03 Summary

This usability study checked that the changes made as a result of the previous study were successful. It also checked the usability of the new elements of the system: the storyboard and run mode. There are still parts of the system that clearly could be improved, particularly the writing recognition. However, students were happy to work with the writing recognition in the form that it had

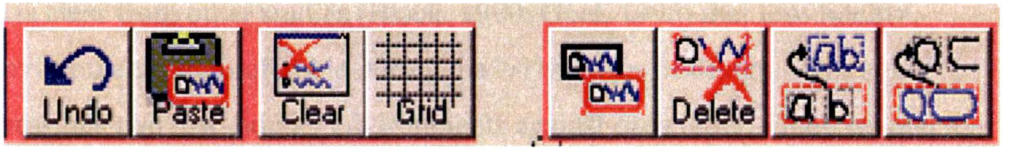
been implemented; they commented that it was very easy to pick up the correct word from the vocabulary and for this reason it wasn't really a problem.

We reinvestigated, without success, the availability of a character recognition module that would integrate with Freeform. We also checked again for alternative digital whiteboard setups, also without success.

We decided to add text to frequently used icons, so as to eliminate any recall problems. The new icons are shown in Figure 55 below.




Mode Icons




Edit Icons

**Figure 55: New Icons With Text**

The results of this study lead us to believe that such a tool should be viable as an integrated component of a programming IDE. We also felt confident that we had a workable platform to test the utility of such a tool for novice programmers. The next chapter describes the evaluation studies we conducted comparing Freeform to a traditional sketch environments and a computer design tool.

# Chapter 8

# Evaluation of Freeform as a Design Environment

In order to evaluate the utility of Freeform as a design environment for novice programmers we conducted a study which directly compared Freeform with a low-fidelity, non-computer equivalent. This exploratory study focused on student use of low-fidelity design as an aid to the initial planning phase of program development.

It is clear from other studies (Black, 1990; Goel, 1995) that low-fidelity environments are preferable during the early stages of design. This study measures two low-fidelity environments against each other: Freeform and a normal whiteboard. The start-point for each design task is a written description of a problem, while the end point is a formal design in the VB IDE.

We particularly wished to compare:

- the quality of the finished designs
- the design process
- the students' understanding of the problem
- the students' preparedness for the next stage of program development
- the students' attitude to low-fidelity prototyping

There are a number of ways that this study could have been conducted. A longitudinal study where students used the software over a number of weeks was considered but it was felt that it would be too difficult to control the variables, also not all the students in a class could participate so there were ethical issues

regarding some students having 'preferential treatment'. We also considered asking individuals, as we had in the usability studies, to complete design tasks but as our initial goal was to provide a group design-space we instead asked students to work in small groups on well-defined tasks.

Each group of students designed two user interfaces: one using the Freeform environment, and the other using a whiteboard and the standard VB6 IDE. During the study we found that groups made more changes in the Freeform environment while checking the design than they did when they were using the whiteboard. To determine whether this was because of the interactive checking we conducted a small supplementary study.

The first section of this chapter (8.01) describes the methodology and details of the study plan. This is followed by an account of the study (8.02), while Section 8.03 reports the data collected. The supplementary study is described in Section 8.04. Section 8.05 summarises the findings and comments on the results.

## 8.01 Methodology

A comparative experiment was used to measure aspects of the Freeform environment against an equivalent traditional environment. In a study such as this, as in one that examines human performance, it is difficult to come to definitive conclusions. However, by isolating the scope of the study, triangulating results from a variety of sources (as suggested by Denzin (1988)) and applying conservative statistical measures, reliable and repeatable results can be expected. Four sources of information were used: participant questionnaires, HCI expert review, the author as observer and an educationalist review. The completed student questionnaires were used to gauge their current practice, opinions on the environments, how well they feel they understand the design problems, and how they view this experience might change their practice and opinions. The design products were evaluated by an independent expert to measure the effectiveness of each process upon the designs. The observations of the author were recorded and the learning environment was reviewed by an educationalist to ascertain whether the different environments affected the learning activity.

Volunteers from an introductory VB programming course, in groups of two or three, participated in the study. Each group was given two interface design exercises: one to be completed in each environment. The details of the problems and environments are set out below. With two problems and two environments there are four possible combinations of problem, environment and order of execution. Each combination was completed by one group of two and one group of three students.

Before the first exercise the students were asked some preliminary questions. They answered another set of questions twice, once after each of the exercises. After the completion of the second exercise they were asked a final question about which, if either, environment they would like to use in the future.

As Freeform is a new tool and the participants had minimal training, an informal approach was taken to the sessions with the author observing and answering questions as required. The sessions were conducted in the Usability Laboratory at the University of Waikato. The laboratory was setup so that the students could work on the whiteboard, computer or LIDS. Four streamed video inputs captured the different work areas and the computer screen (Figure 56).

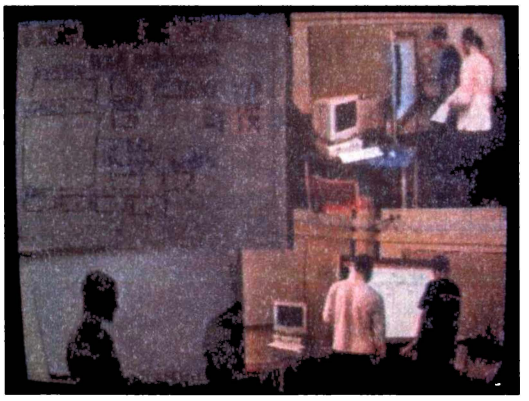


**Figure 56: Video Capture of Evaluation Study**

### 8.01.1 The Environments

Our goal was to isolate the affect of the Freeform software by minimises the differences between Freeform and the other environment. In the 'traditional' environment we provided both a normal whiteboard for informal design and a

standard computer running the VB IDE for the form creation. The second environment involved the use of LIDS hardware with Freeform software. With this arrangement the main difference was the Freeform software with its inherent support for editing and interactive checking.

In the traditional environment students were required to sketch their design on the whiteboard and use the scenarios provided to check the sketch. When they were confident that the design was correct they made the VB form on the computer (Figure 57). They were free to make changes when transposing from the sketch to the form but were not instructed to recheck the VB form.

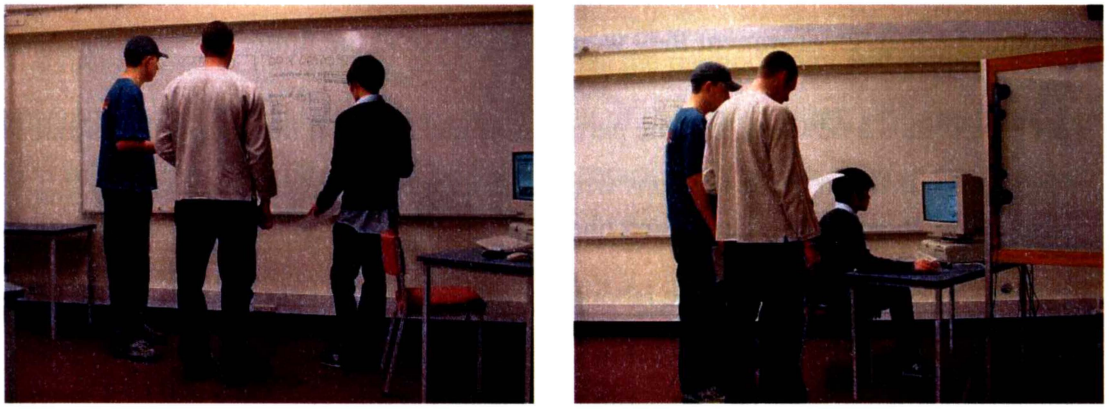


**Figure 57: Group at Whiteboard and Then Computer**

In the Freeform environment students sketched their design on the LIDS whiteboard, using the Freeform software. They then check it with the scenarios provided in the Freeform run mode, after which they used the Map and Make facilities to translate the sketch to a VB form (Figure 58).

**Figure 58: Group at Lids**

### 8.01.2 The Design Problems

The second usability study (Chapter 7) suggested that it would take students two-to-three hours to design two multi-form interfaces. Most of the points we were interested in could be demonstrated with the tackling of simpler problems. Consequently, two roughly equivalent problems were used in the study; both are simple single-entity record forms, of similar complexity and size. A brief description of the problems is given below. A complete copy of the problem descriptions and scenarios, as given to the students, is included in Appendix 3.

**Problem 1 – Book Catalogue Form**

The students were required to design an interface for a book catalogue. This was to include fields for: the title, the names of authors, ISBN, the year of publication, the genre, size and the binding. The problem statement described that there could be one or more authors for a book and that it is important not to record the name of an author more than once. Other areas where decisions were required involved: the ISBN which has four sections; the problem definition, which implies that the publisher and genre should be able to be chosen from a list; the size, which has figures for both the book's height and width; and the binding, which is described as hardback/paperback.

Two scenarios were provided for checking the design. These scenarios incorporated some of the expected variations with, for example, one having one author, the other two, and different numbers of digits in the different sections of the ISBN.

**Problem 2 – Dog Registration Form**

The dog registration form had two sections, one for the owner and the other for the dog. For the owner the form required: a name, address, date of birth and phone number. For the dog the requirements were: a location if different to the owner's address, a name, colour, breed, sex, age, its special purpose status and the fee payable. The problem statement explained that the special purpose status was for dogs, such as guide dogs, which do not pay fees. Additionally, the problem statement stated that the program calculates the fee owing as dependent upon the information that has been provided.

The two scenarios again included some of the expected variations; different length addresses, one dog with a different address to its owner and one which referred to a special purpose dog. The colours and breeds were familiar so as to imply that these might be chosen from a list.

### 8.01.3 Design of Experiment

The four different combinations of environment and problem were each completed by two groups. The combinations of environments and task presented to each group is set out in Table 6 below

| | Traditional Environment | | Freeform Environment | |
|---|---|---|---|---|
| Group | Task Order | Problem | Task Order | Problem |
| 1 | 1st | Book Catalogue | 2nd | Dog Registration |
| 2 | 1st | Book Catalogue | 2nd | Dog Registration |
| 3 | 1st | Dog Registration | 2nd | Book Catalogue |
| 4 | 1st | Dog Registration | 2nd | Book Catalogue |
| 5 | 2nd | Dog Registration | 1st | Book Catalogue |
| 6 | 2nd | Dog Registration | 1st | Book Catalogue |
| 7 | 2nd | Book Catalogue | 1st | Dog Registration |
| 8 | 2nd | Book Catalogue | 1st | Dog Registration |

**Table 6 Combinations of Problem and Environment**

Each group in the study followed a similar procedure. After completing the ethical approval form, they were given a questionnaire each and were asked to complete the first three questions (Table 7). Before completing the task in the traditional environment they were asked to

"Sketch a design for the form on the whiteboard, draw a form border and all of the required controls, visually indicating the control types by the shape and size of the objects, static labels and captions are indicated by writing the word on the design"

and then:

"Check your form with the scenarios provided. Check that you have a control for each data item and that each control is of the appropriate type and size."

After checking the design and correcting any design problems the group created the form with the VB form builder to the stage where they were ready to write the code (they were not required to name the controls to save time).

After the completion of this exercise they were given time to fill in the next section of the questionnaire (Table 8).

Before commencing on the Freeform task groups were shown how to use the Freeform environment. The training covered the pen, drawing, writing, editing, shape and letter recognition, and the run mode. Information on translation to a VB form was left until required. The group was then asked to:

"Sketch a design in the Freeform environment following the instructions you have been given. Focus on the design, while remembering the fundamentals required for recognition"

and then:

"Use the Freeform play mode to check your form with the scenarios provided. Check that you have a control for each data item and that each control is of the appropriate type and size"

When the design was completed and any problems corrected, the group was shown how to invoke recognition and conversion to a VB form. When the form was completed to their satisfaction, they answered the set of post-exercise questions for the exercise (Table 8).

After both tasks were completed the participants answered one final question on which environment they would prefer to use if both were available to them (Table 9).

### 8.01.4 Data Sources

The students, as individuals, completed the questionnaire in four stages. At the beginning of the session, they answered questions on their prior experience with whiteboards, their current habits for sketching designs and their beliefs in relation to the value of sketching designs (Table 7).

| | | | | | | |
|---|---|---|---|---|---|---|
| I use a whiteboard for planning task | Always | Usually | Sometimes | Rarely | Never | |
| I hand sketch interface designs before I create them in the programming IDE | Always | Usually | Sometimes | Rarely | Never | |
| I think sketching an interface design is necessary when planning a program | Strongly Agree | Agree | Neutral | Disagree | Strongly Disagree | |

Table 7 Pre-Questions

After each exercise the students answered a set of questions about their experience. The same questions were used for each exercise (Table 8).

At the completion of the session a final question asked which environment the students would prefer to use for informal designs: a whiteboard, Freeform or neither (Table 9).

The completed VB forms were assessed by an HCI expert to evaluate the affect of the environments on the final product. The reviewer was not given any information about how the designs were created or which group created them. He was asked to look at the general layout of the form, the choice of controls and general appropriateness of the design. Each design was given a mark out of ten.

The video tapes of the sessions were reviewed by an educationalist who commented on the learning activity. He looked at the group interactions, the learning cycle, and the different tasks or approaches taken in the different environments.

The facilitator both conducted and observed the sessions, noting the design activities that were taking place, in particular, changes of direction and discussion about alternate strategies and any difficulties experienced with the tools.

| | Strongly Agree | Agree | Neutral | Disagree | Strongly Disagree |
|---|---|---|---|---|---|
| General Questions | | | | | |
| This exercise was enjoyable | | | | | |
| This exercise has helped me to understand the software design process | | | | | |
| This exercise motivated me to find out more about programming | | | | | |
| About the problem | | | | | |
| I understand the problem | | | | | |
| This environment helped with my problem understanding | | | | | |
| I feel prepared to complete the program | | | | | |
| About the environment | | | | | |
| Creating the sketch was easy | | | | | |
| Checking the scenarios was easy | | | | | |
| I would like to use this method of program planning in the future | | | | | |
| This experience has made me value sketching interface design more | | | | | |

**Table 8 After Each Exercise Questions**

| |
|---|
| Given a choice I would like to do my informal design with<br>**whiteboard** ☐ **Freeform** ☐ **none** ☐ |

**Table 9: Post Questions**

# 8.02 Study

The study was completed by eight groups, groups 1,3,5, & 7 consisted of three people while groups 2,4,6, & 8 consisted of two people. Groups 1 & 2, had the same combination of task order and problem as did, respectively, 3 & 4, 5 & 6, 7 & 8. Figure 59 shows a number of sample sketches and forms from the whiteboard and Freeform.

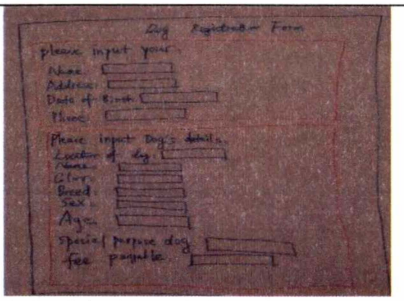
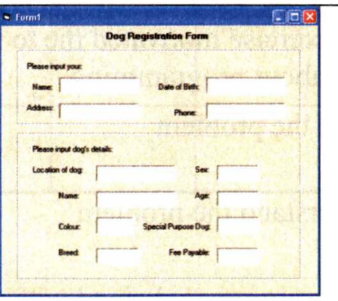
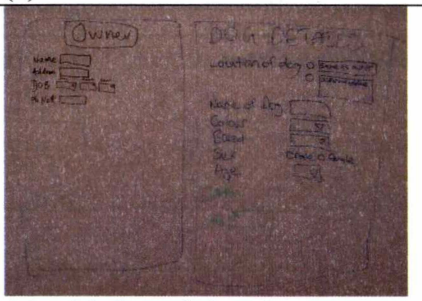
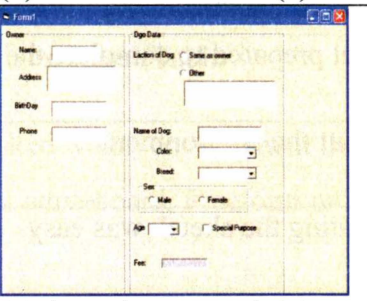
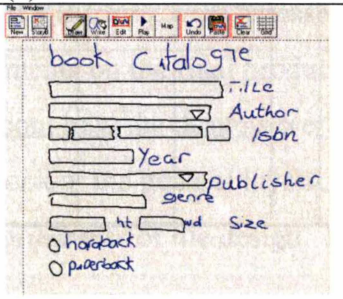
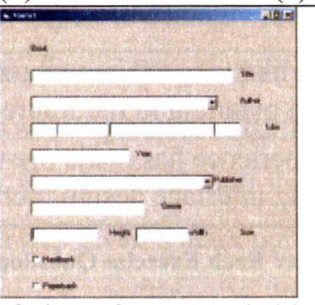
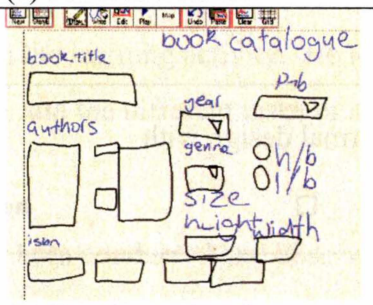
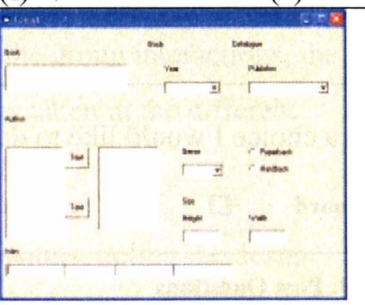
<table>
<tr><td>Group 2.2</td><td>  <br>(a) whiteboard sketch</td><td>  <br>(b) form from sketch (a)</td></tr>
<tr><td>Group 3.2</td><td>  <br>(c) whiteboard sketch</td><td>  <br>(d) form from sketch (c)</td></tr>
<tr><td>Group 3.1</td><td>  <br>(e) Freeform sketch</td><td>  <br>(f) form from sketch (e)</td></tr>
<tr><td>Group 3.2</td><td>  <br>(g) Freeform sketch</td><td>  <br>(h) form from sketch (g)</td></tr>
</table>

**Figure 59: Selection of Sketches and Forms**

## 8.03 Findings

The findings from this study are presented in four parts: in the questionnaires, an evaluation of the interface designs, the observations of the author and in an evaluation of the learning experience.

### 8.03.1 Student Questionnaire

A total of twenty individual questionnaires were completed, this consisted of four groups of three and four groups of two (one group of three and one group of two did each combination of environment/problem/sequence). The raw data are presented in Appendix 3.2. The responses were coded numerically with the most left column (always or strongly agree) coded as 1 to the most right column (never or strongly disagree) coded 5.

The three initial questions (Table 7) asked about the students' current practice and beliefs. Most rarely or never (18) used whiteboards for planning. There was a wider range of practice for pre-sketching interface designs: six usually sketched first, eight sometimes sketched and the remaining six rarely or never pre-sketched their designs. The last question in this section asked about their beliefs in the necessity of sketching interface designs; most (thirteen) agreed that it was necessary, with three strongly agreeing and four giving a neutral answer.

The participants answered the next set of nine questions twice; once after each exercise. As we were looking for the differences between the two environments each question was compared for the two exercises. The statistical analysis was an ANOVA analysis of variance test. This test allowed us to compare the differences in the two environments while taking into account the order of task completion and the two different problem scenarios that were used. The mean score for *each group* because there is normally some *group think*, this is a more stringent test than using individual scores. Using this rigorous approach to statistical variance implies that significant results maybe assumed repeatable with confidence.

The results highlight the affect of environment, the affect of order of completion and the affect of the problem scenario. The problem scenario did not affect the

answers to any of the questions, however the order of completion did affect one question, *'Creating the sketch was easy'*.

The analysis for the question '*I understand the problem*' was particularly interesting in that, although the students found the second problem easier, they found was much easier if they did using Freeform. When Freeform was used for the first exercise, the means were 2.0 and 1.8 for Freeform and the traditional environment respectively. When the traditional environment was first the means were 1.90 and 1.30 for the traditional environment and Freeform respectively. This gives a statistically significant result for the environment for this question. This question is used as an example of the analysis undertaken for each question in Appendix 3.2. Table 10 shows the results for the environment affect for all questions, the session affect for *'Creating the sketch was easy'* and the session and environment affect for '*I understand the problem*'.

Freeform scored at a significantly higher level ($p < .05$) for the questions:

- This exercise was enjoyable
- This exercise motivated me to find out more about programming
- I would like to use this method of program planning in the future
- This experience has made me value sketching interface designs more
- I understand the problem

The following two questions were higher for the Freeform environment ($p < .10$)

- I feel prepared to complete the program
- Checking the scenarios was easy

One question was statistically higher for the second problem each group did, regardless of problem or environment.

- Creating the sketch was easy

Two questions showed no statistically significant difference:

- This exercise has helped me to understand the software design process
- This environment helped me with my problem understanding

Analysis of Questionnaire Data by Environment

| Questions | Means | | Degrees of freedom | s.e.d | P value |
|---|---|---|---|---|---|
| | Free-form | White-board | | | |
| This exercise was enjoyable | 1.40 | 2.20 | 4 | ± 0.12 | 0.003 |
| This exercise has helped me to understand the software design process | 1.75 | 2.15 | 4 | ± 0.20 | n.s. |
| This exercise motivated me to find out more about programming | 1.50 | 2.50 | 4 | ± 0.24 | 0.015 |
| | | | | | |
| I understand the problem | 1.65 | 1.85 | 4 | ± 0.13 | n.s.† |
| This environment helped with my problem understanding | 1.80 | 2.20 | 4 | ± 0.21 | n.s. |
| I feel prepared to complete the program | 1.75 | 2.15 | 4 | ± 0.18 | 0.098 |
| | | | | | |
| Creating the sketch was easy | 2.10 | 2.00 | 4 | ± 0.16 | n.s.‡ |
| Checking the scenarios was easy | 1.90 | 2.55 | 4 | ± 0.29 | 0.091 |
| I would like to use this method of program planning in the future | 1.70 | 2.35 | 4 | ± 0.24 | 0.019 |
| This experience has made me value sketching interface design more | 1.35 | 2.00 | 4 | ± 0.20 | 0.032 |

‡ Analysis of Questionnaire Data by Session

| | | | | | |
|---|---|---|---|---|---|
| Creating the sketch was easy | 2.35 | 1.75 | 4 | ± 0.16 | 0.020 ‡ |

† I understand the problem interaction of session and environment

| Session | Freeform | Whiteboard | Degrees of freedom | s.e.d. | p |
|---|---|---|---|---|---|
| 1st | 2.00 | 1.80 | | | |
| 2nd | 1.30 | 1.90 | | | |
| | | | 8 | ± 0.26 | 0.038 |

**Table 10: Summary of Questionnaire Analysis**

We also compared student opinions, in the first section of the questionnaire and after each exercise, on the necessity of sketching. Most people scored the pre-question '*I think sketching an interface design is necessary when planning a program*' quite highly (mean 2.05). The post-exercise question asked whether they now valued sketching *more*. We analysed their change in opinion between these two questions and found that Freeform gave a significant boost to their opinion of sketching, with the mean change for Freeform being 0.70 and the mean change for the whiteboard being 0.05 (see Appendix 3.2 for details). Figure 60 shows each individual's score for these two questions for Freeform and the whiteboard plotted against their pre-question response. There is a clear clustering of the post-questions for Freeform with the strongly agree response where the whiteboard clusters around agree.

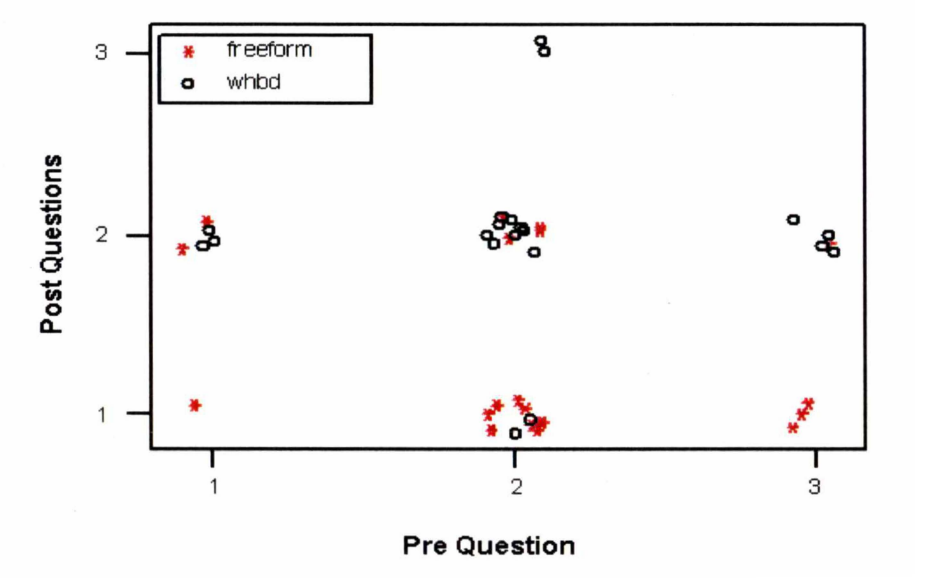



**Figure 60: Value of Sketching Before and After Exercises**

(points are randomly offset to convey density)

The final question asked the students to nominate their preferred design environment: whiteboard, Freeform or neither. Seventeen people chose Freeform as their preference, one preferred the whiteboard and two neither.

### 8.03.2 Designs

The designs were 'marked' by a computer science HCI specialist, Dr Matt Jones. He looked at the choice of controls and general layout. Five groups scored a better mark for their form designed in the Freeform environment, one score the same mark in both and two scored better marks for their form designed in the traditional environment. These results are not statistically different (see Appendix 3.3 for details).

### 8.03.3 Observations

We noticed that the students made many more changes in the Freeform environment; we reviewed the video tapes to quantify this observation. The mean number of changes in the traditional environment was 1.13, while in the Freeform environment it was 3.5 ($p < .01$). No group made more changes in the traditional environment than in the Freeform environment. This gives a 95% confidence interval for mean difference (Freeform – traditional environment) of 0.59 – 4.16. A summary of these values is contained in Appendix 3.4.

We asked the students if they had noticed that they had made more changes in the Freeform environment and if they had, if they knew why. Some were not really aware of the difference, however those that were suggested that this was because they were more into the computer way of thinking than they were when working on the whiteboard.

Most of the changes in the Freeform environment were made after the students had used the run mode where they had interactively checked the designs with the scenarios. On the whiteboard they would look at the scenario and look at the design, generally they not try to use the design (although two groups who used Freeform first did fill in parts of the design on the whiteboard). A typical example of a change that was made after trying the scenarios was the controls for the dog registration address data. The scenarios had two address lines. Most groups initially drew a single line text box to hold the address. In the traditional environment, the finished form of three of the four groups had a single line text

box. In the Freeform environment, three out of the four groups created space for multiple address lines (either a multi-line text box or multiple text boxes).

This raised the question as to whether the greater numbers of changes in Freeform was due to the interactive checking. A supplementary study was conducted to isolate this particular aspect; this study is explained in Section 8.04.

Although we timed the exercises, they were not strictly comparable as the Freeform session involved some training which was interspersed with the designing. The whiteboard exercise took about 25 minutes where the Freeform exercise took about 35 minutes. The time for Freeform exercises could be reduced if they both did not include training and the users had more experience with the environment.

We observed discussion about the functional requirements of the programs in both environments. The quality of sound recording was not such that any analysis of this could be undertaken, but our impression was that the run mode encouraged more discussion on functionality than did the static whiteboard checking.

There were a number of small usability issues that were uncovered during this study. The ink in run mode is cleared each time the users exit the run mode. The run mode ink would be better left and cleared only by user action. The align to grid that is done during the map process can result in glyphs overlapping each other and although the VB form is much improved, as a result of this alignment process, the sketch ends up looking worse. One possible solution to this would be to superimpose a tidy design with regular shapes over the sketch at map time leaving the underlying sketch as it is. Also handwriting is usually larger than print. As a consequence, the VB form is to spread out horizontally. Modification of the software would address these issues.

### 8.03.4 Learning Environment

An educationalist with an interest in learning environments (Dr Anthony Morrison) reviewed the videos of the sessions. He felt that the groups worked in a similar way in both environments. However, he had some suggestions as to why the students made more changes in Freeform.

Dr Morrison proposed that in addition to the unfinished look of a sketch (Wong, 1992) the flexibility of the Freeform environment encouraged change because it is possible to add, delete and move elements more easily than the equivalent whiteboard environment. Freeform also provides a feeling of security, in that any change can be undone.

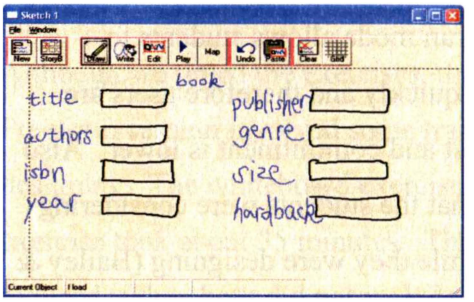
Dr Morrison also suggested that the integrated run mode allows students to complete the learning cycle (Kolb, 1984) more quickly and therefore users are more prepared to change their design, as the cost and commitment is lower. Also for the activity that was going on, it was clear that the students were considering the behavioural requirements of the program while they were designing (Bailey & Konstan, 2003; Gross, 1998; Stahovich, 1998). He concurred with the author's observation that there appeared to be more of this type of activity when the students were working on Freeform.

## 8.04 Supplementary Study

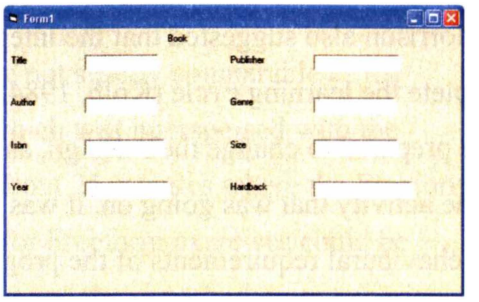
We conducted a small supplementary study to determine whether the additional changes we observed in Freeform were because of the interactive checking or the sketch representation. We had noted that while groups filled in the design in Freeform they did not do this on the whiteboard. It seemed that most of the changes that were made, were made as a result of trying to use the form. We designed a small study to count the number of changes resulting from interactively checking designs rendered as a sketch (Freeform) or a formal design (VB form). We defined a change as adding or deleting a control/glyph, changing the type of control/glyph or moving a control/glyph to a different part of the design. Activities like aligning and sizing were not considered to be changes.

Two simple problems were used: the book catalogue that was used in the earlier study and a credit application form. We created two renderings of each problem: one a Freeform sketch and the other a VB form (Figure 61). Although these designs look unrealistically simple, (just labels and text boxes), in the previous study many of the groups started their designs like this.

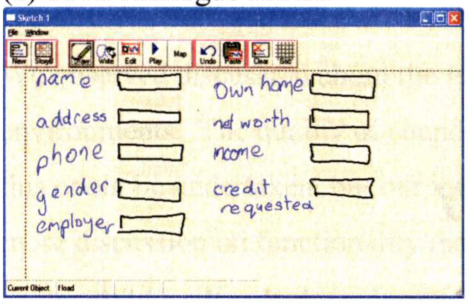
Small groups of students were asked to check two designs each; one rendered as a sketch and the other as a VB form. They were all familiar with VB, but none had used Freeform, so they were given basic training in drawing, writing, editing and run mode (we did not ask them to convert the sketch to VB). Some groups checked the VB form first, others checked the sketch first.



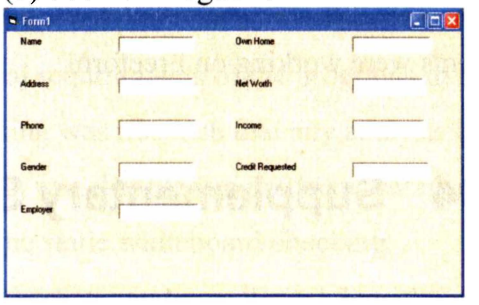

(a) book catalogue sketch




(b) book catalogue form




(c) credit application sketch




(d) credit application form

**Figure 61: Designs Used in Supplementary Study**

Six groups of two or three students participated in this study. They were given the same instructions for each exercise.

"Use the problem statement and scenarios provided to check the design you have been given. As part of this process, try to fill the form in with the scenarios. Change the design to provide a better interface for the problem"

This small study produced very interesting results. As with the first evaluation study we observed more discussion about functional requirements when working on Freeform. Also of the six groups, five groups made more changes in Freeform, regardless of the problem or the order of the exercise. The other group made the same number of changes on both designs, when they did the VB exercise, they decided to hand-sketch the design before changing it on the computer. The mean number of changes made in VB was 6.5 and in Freeform 8.67. This gives a 95%

confidence interval of mean difference 1.03-4.96 (p 0.01). A complete table of changes is shown in Appendix 3.5.

There were some clear differences between the changes that were made to the sketches and formal diagrams. We will illustrate this using comparable elements; each problem include an item that would best be represented as radio buttons and another that would best be represented as a dropdown list.

The book binding, on the book catalogue form, and gender, on the credit application form, would best be expressed as mutually exclusive radio buttons. On the informal sketch all six groups changed these elements to radio buttons; on the formal design four groups used radio buttons, one group used checkboxes, while the other group did not change the design.

With the book catalogue problem the genre would usually be a dropdown list as would the income categorises on the credit application. There was a similar pattern to the changes here; with the informal designs all groups changed the elements to a dropdown list, while with the formal designs, four groups changed to dropdown lists, but the other two groups made no change. There was no relationship between the groups and the changes; each 'error' with the radio buttons and lists was made by a different group.

Most groups were surprised when they were told that they had made more changes in Freeform. We noticed that they spent a lot of time in VB doing *busy* work, aligning, sizing etc (Black, 1990; Goel, 1995), where in Freeform, although they were not familiar with the environment, they spent more of their time designing.

## 8.05 Summary and Discussion

The first evaluation study compared Freeform to a traditional alternative and the second Freeform to a high-fidelity, computer-based alternative. The first study collected information from four different sources: the study participants, an HCI expert, an educationist and the author, who observed all the sessions. The second study looked specifically at the number of changes made in the different environments. This discussion is structure around the main questions stated at the beginning of this chapter.

### 8.05.1 Quality of the Finished Designs

The groups in the first study created slightly better designs in Freeform and the design products in Freeform from the supplementary study were definitely superior. Earlier studies that have compared low-fidelity, traditional and high-fidelity computer-based environments have consistently found that the low-fidelity, traditionally environments were better from a design perspective (Bailey & Konstan, 2003; Black, 1990; Goel, 1995). This outcome is considered to be so because of the constraints and distractions created by computer-based tools. There are still some constraints in the Freeform environment: the need to draw in a way that can be recognised and the different inking modes. We postulate that these should adversely effect the design products. However, this seems to be counter-balanced by the advantages of interactive checking. We believe that design products that are, at a minimum, as good as those created in a traditional low-fidelity environment is a very positive result.

### 8.05.2 The Design Process

The interactive checking in Freeform provides a new dimension to the design process by automating the 'play computer' suggested by Rettig (1994). This checking resulted in more changes than either checking low-fidelity designs on the whiteboard or checking high-fidelity designs interactively in VB. Almost all of the changes we observed, regardless of environment, improved the design. This is consistent with the student questionnaire results that checking the

scenarios was easier in Freeform. This implies that interactively checking sketches on a computer-based tool produces better designs than either of these alternatives.

The Freeform environment is effective at supporting the design process. It makes changing the sketch easier by providing the editing capabilities expected of computer tools. Also the integration into a programming IDE means that designers can move more easily from informal to formal designs.

### 8.05.3 Students' Understanding of the Problem

The questionnaire results would suggest that Freeform had a bearing on students' understanding of the problem. In addition both the more detailed discussion on functional requirements that were observed when the students were working on Freeform and the superior designs created on Freeform during the supplementary study would suggest that the students had a better understanding of the problem.

Contrary to this the questionnaire results suggest that the environment had no effect on the students' understanding of the design process or that the environment itself helped them to understand the problem. Further studies may clarify this point.

### 8.05.4 Students' Preparedness for Program Development

Preparedness for the next stage of program development should be closely linked to problem understanding. The questionnaire results suggest that the use of Freeform resulted in a higher level of preparedness. Clearly a longer study which required students to go on and complete the programming task would provide better evidence of the effect of the initial design task as a forerunner to program coding.

### 8.05.5 Students' attitude to Low-fidelity Prototyping.

Students had a positive attitude to the experience. They overwhelmingly said they enjoyed using Freeform more than the whiteboard and said they would like to use it again. It also made them value sketching more and it motivated them to find

out more about programming. It is possible that the increased motivation to learn about programming has little to do with the tool's usefulness but was because the students were aware that the software was developed in VB by the author. However, we can conclude that, this aside, it was a positive learning experience for them.

The outcomes from this study indicate that informal computer-based design tools are likely to be useful for novice programmers. We found that they enjoyed the experience, the designs were comparable to those created in a traditional environment, the tool gave both them the support they expect of a computer tool and integrated informal design into a standard program development environment.

# Chapter 9
# Review, Conclusions and Future Work

This thesis has explored the utility of a computer-based low-fidelity environment to support group user interface design activities. As such, it focused on supporting students who are learning to program. To this end, we have developed and evaluated a tool to meet these specific requirements.

From the literature on design, and what we know about learning to program and other sketch tools, we established that the critical factors for such a tool to be successful are that it retains the look and feel of a whiteboard, provides a group-space, supports editing and storage, supports interactive checking and integrates into a program development environment. These fundamentals guided the development of Freeform. A review of the software is given in Section 9.01.

Having developed Freeform, we then used the software to explore how students might use such an environment as part of the software development process. Of particular interest was the design process, the quality of the designs produced, the student's understanding of the problem they were endeavouring to solve, and their attitude to low-fidelity prototyping and Freeform. Section 9.02 comments on of our evaluation studies from the perspective of the program design process.

Section 9.03 discusses this work in the context of our initial goals and draws some conclusions on what has been achieved, and what the contributions this work has made to the field of software for user interface design. Finally, Section 9.04 offers suggestions for future work.

## 9.01 Review of Freeform

The LIDS setup that we used throughout this project provided a shared-space (whiteboard) where users could sketch directly onto the output surface. This surface has the advantage of being simple and low-cost, however the Mimio pens are a little cumbersome, and lack the means to change modes and passively track pen movement. We have designed around the constraints of LIDS; interaction requirements for digital whiteboard software will differ depending on whether passive pen tracking and mode changes are possible.

The sketch-space in Freeform has emphasised in-place inking and endeavoured to maintain the look and feel of a whiteboard. The comments from users in the usability studies such as the pen is 'easier than a mouse' suggest that we have succeeded in doing this. While it is possible for users to draw and write anything, anywhere, just as they would on a normal whiteboard, there are constraints required to enable the software to convert the sketch to a formal diagram. The most disruptive of these is the two inking modes; a single inking mode would be preferable and is likely to be possible as more sophisticated recognition engines become available with more suitable application program interfaces (API).

There are two approaches to supporting editing interaction in pen-based environments: gesture or button/menu-based editing. Other researchers have supported gesture-based editing (for example Landay & Myers, 2001). We have found that there are problems with gesture-based editing because, it is unfamiliar to the users, notwithstanding that when recognition fails there is confusion and extra work is required to bring the sketch to the desired state. We provided two gesture-based functions: a delete gesture and overwrite. The undo button was frequently used during the studies. Other editing tasks require the user to change to edit mode, select the ink for revision and then move or resize.

The ability to interactively edit is something we take for granted in computer environments and is an advantage over traditional environments. The first Freeform prototype had minimal editing support, and we added, after receiving user feedback, more editing support to the second prototype. We believe that we achieved a good balance between keeping the feel of a whiteboard and providing

the functionality expected in a computer-based environment. The LIDS setup required us to rethink how editing should be accomplished because pen movement is not passively tracked and the screens have a small parallax error which together makes it more difficult to grab the handles.

The storyboard shows a thumbnail of each sketch. Here, the user can move sketches around the fixed display spaces and create navigation links between sketches. We were able to provide a non-modal interface as the software can disambiguate the user's intentions. No difficulties arose with the storyboard during the studies, however we can see possibilities for extending this view. These views are presented in the section on future work.

The run mode proved to be one of the most beneficial parts of the tool. The ability to interactively check sketched designs with scenarios showed benefits in both the main and supplementary evaluation studies. Others have provided more behaviour, for example Landay & Myers (2001) had scroll bars that moved up and down and Bailey et al. (2001a) included the ability to play media (video, sound etc) for their multimedia design tool. We have not included any sketch-based behaviour because it requires the sketch to be fully recognised and this is contrary to our intention to downplay recognition during design. Additionally, it must be added that the affect of sketch behaviour on the design process is not understood.

Recognition of ink data, particularly character recognition, is technically challenging. Freeform implements Rubine's (1991) relatively simple pattern-matching algorithm; it gives satisfactory results for shape recognition, but is inadequate for character recognition. Character recognition was the most requested enhancement in the first usability study, a result that is consistent with comments from Bailey & Konstan's study (2003). We have had limited success with word recognition by both adding to Rubine's feature set and matching words against a vocabulary. This approach provided enough support to encourage students to write. Moreover, they had no problems with choosing the correct word from the vocabulary list when the recognition failed. The comments from our first usability study and Bailey & Konstan's (2003) study, together with the outcome of our second usability study, show that character recognition is important to user-acceptance of such a sketch tool.

A rule-based approach is used to join strokes and words together to make glyphs. This worked well in the test system. The libraries of letters, shapes, and rules are exposed to the users. Although these interfaces were not tested in the studies they are an important first step in providing a user-configurable sketch system.

When the user taps the *map* button recognition is completed and all the glyphs are snapped to the grid. Recognition information is shown on the sketch as labels over each glyph. The user can then correct any misrecognition by tapping the label and choosing the correct control type or word. The correction process worked well during the trials and the snap to grid resulted in a significant improvement in the quality of the VB forms. However, the software moved the ink onto the grid spoilt our original intention of leaving the image alone. An alternative approach is suggested in the section on future work.

The VB forms from the first prototype were untidy and the second most frequent request for improvement to the tool was for a tidier VB form. This desire for tidiness of the formal design is consistent with the comments of Wong (1992) and reinforces the underlying unsuitability of high-fidelity tools for early design. Creating a satisfactory VB form from the sketch proved to be challenging. While we have made significant progress we can see opportunities for further work, which is described in Section 9.04.

## 9.02 Computer Supported Low-fidelity Design

The design process is one of imagining, expressing, exploring and refining. Imagining is the internal process, which when expressed as a sketch, can be used as a prop to explore, communicate and refine the design (Tversky, 1999).

Expressing initial designs is best accomplished in a minimally constrained environment. With Freeform we have balanced the need for computer-supported editing and recognition with this need for a minimally constrained environment. The evaluation study described in Chapter 8 directly compared Freeform to a normal whiteboard. We found the designs were of similar quality. This is a

pleasing result as the studies of Black (1990) and Goel (1995) compared traditional low-fidelity environments with computer-based high-fidelity environments and found that the traditional environments were preferable.

Exploring in the context of user interface design is better supported on a computer than with a static design because of the ease of interaction. By using scenarios to check the interface design students are likely to have a better understanding of the problem and create a more appropriate interface (Rettig, 1994). When novice programmers are attempting to solve simple problems having appropriate user interface components and a good understanding of the problem before they start to program is important.

In the first evaluation study, the students made more changes to their designs when checking them on Freeform and there was a statistically significant increase in the student's self-evaluation of problem understanding when they used Freeform for the second task. The supplementary study looked specifically at the number of revisions made as a result of checking low-fidelity and high-fidelity designs interactively; there were significantly more substantive changes to the low-fidelity designs. When working with the high-fidelity design students spent a great deal of time aligning and sizing controls while making fewer substantive changes. This was in contrast to their experience on Freeform where there was very little *busy* work. This is consistent both with Wong's (1992) claim that high-fidelity designs focuses too much attention on detail and with Goel's (1995) observations of there being more radical changes in low-fidelity design and more refinements with high-fidelity designs. This suggests that Freeform encourages focus on the design task rather than presenting distracters such as recognition or alignment.

There is a slight contradiction in the evaluation study results. The groups made more revision on Freeform, which should result in the Freeform designs being better than the whiteboard designs. The Freeform designs were a little better, but not at a statistically significant level. It may be that a larger study would find a difference or perhaps the constraints of Freeform, noted in 9.01 above, slightly impair the first rendering of the design, but this is countered by a better checking environment.

We noted that students talked about the functional requirements of the programs both in the usability studies and in the evaluation studies. In the main evaluation study questionnaire, students rated their problem understanding and their preparedness to complete the problem as higher with Freeform. This result is similar to that achieved in Bailey & Konstan's (2003) study where participants ranked Demais significantly higher than the other two environments for communicating behaviour. Although it was not possible to judge the actual problem understanding of our study participants, we suggest that Freeform is likely to help the student's problem understanding because the interactive checking encourages them to think about user interface and behaviour requirements.

The final phase of design, when using an IDE, is the creation of the form in the IDE form builder. For this to be automatic, the sketch must be recognised. We deliberately hid recognition during the earlier stages of design so as not to interrupt the design process. Freeform provides both recognition and beautification to support the process of converting a sketch to a formal design.

## 9.03 Discussion and Conclusions

Freeform has contributed to the knowledge of what is possible and desirable in a sketch environment. The usability tests that were undertaken on the LIDS environment have added to the understanding of how design can be supported on digital whiteboards. In particular, the integration of Freeform into the IDE provides a computer supported environment for programming from early design to finished product. The inclusion of character recognition has demonstrated its value. Also the user's dissatisfaction with the untidy forms of the first prototype led us to beautify the forms, the value of which the subsequent studies showed.

The first evaluation, study reported in Chapter 8, shows similar results to those of Bailey and Konstan (2003). Bailey and Konstan's study involved the use of practicing multimedia designers while ours used novice programmers. The similarity of results indicates that the requirements and utility of computer-based sketch tools is likely to hold true, just as it does for traditional tools, across a

range of disciplines. The supplementary study was the first to directly compare high and low fidelity computer-based environments for interactive checking. The results from this study suggest that, for interactive checking, computer-based low-fidelity sketches retain their advantages over high-fidelity designs.

User interface builders are similar a cross a wide range of IDEs and database tools. We chose to integrate Freeform with VB6, which created some technical difficulties because form creation is deeply imbedded in the IDE (Marsden, 1997). Other IDEs such as Delphi™ or any of the Visual C environments could have been used and may have offered easier integration. The requirements from a design perspective are very similar across these types of products and we foresee no difficultly in extending the building blocks in Freeform to other IDEs.

Our usability and evaluation studies focused on novice programmers. Experienced software developers face the same challenges of both needing to understand the problem and designing a usable interface. Clearly the problems that professionals are dealing with are larger, more complex and less well defined than those given to novice programmers. Wong (1992), a professional designer, went to some trouble to create interactive sketches to use as a discussion medium with clients by scanning hand-drawn designs and then using software to overlay the designs with navigation links. In doing this she created the type of prototype that we have developed in Freeform, the latter being quicker and simpler of the two. Freeform is likely to be a useful addition tool for scenario-based software development methodologies (Carroll, 2000).

From a wider perspective, some design environments such as CASE, are quite similar in that the end product is a diagram. But others have quite different requirements, for example CAD tools model 3D spaces. Other disciplines have developed techniques for rendering complexity, for example a layered approach is the norm in architecture (Trinder, 1999). It is clear from our research that design principles hold true across other disciplines. These results, together with results of Bailey and Konstan's (2003) study lead us to believe that our observations about the value of low-fidelity prototyping with computer support will hold true in other design disciplines.

## 9.04 Future work

The Freeform tool has had two iterations of development and usability testing, as well as having been used in an evaluation study to examine the utility of a sketch tool for novice programmers. This section describes the ways that we see the Freeform tool could be further developed, and suggests both usability and evaluation studies that would add to the understanding of informal prototyping of designs using computer-based sketch tools.

From a technical perspective the biggest outstanding challenge is to provide a single mode sketch-space which seamlessly recognises drawing and writing. Microsoft Tablet XP ™ operating system has introduced ink as a fundamental data class; this may offer a way forward with this problem. The recognition of shapes could also be improved by using more complex algorithms and incorporating Long's (1999; 2000) ideas on designing unambiguous gestures. To provide extensible recognition library interfaces are important; Gross (1994) included an effective method of describing glyphs by example, which is more user-friendly than the text interface included in Freeform.

We initially planned to support two-way transfer and synchronisation of informal sketches and formal VB forms. This proved to be technically difficult and has not been implemented. However, it would be useful to be able to change a VB form and have that change reflected in the sketch.

The storyboard could be more flexible. The thumbnails in Freeform are fixed in position and size, and it may both be better for these to be resizable and to be able to be repositioned on the storyboard so that the sketches are in a pattern that reflects the relationship between them.

The current version of Freeform moves ink onto a grid when the map button is pressed. As suggested earlier, we now think it would be better to leave the ink where it is and overlay the sketch with regular shapes that indicate where the VB controls will lie. As part of the form beautification, it maybe useful to examine the relationships between glyphs so that they are set out in vertical and horizontal

lines. This would also reduce the horizontal spacing where hand-written words take more space than the text equivalents require.

There are large parts of this project which are directly transferable to both other IDEs and similar types of design environments, such as CASE tools. There is potential to package parts of the tool up and expose interfaces for the parts that need to be configured for individual application integration.

There have been very few formal evaluations of sketch systems. We see further studies being useful both for the improvement of the usability of sketch-spaces and to discover more about the design process, as supported by these types of tools. The usability aspects of sketch-spaces are determined by both the hardware and software capabilities. We expect there to be advancements in digital whiteboards that will affect the interaction requirements.

The usefulness of editing gestures in a sketch environment should be more formally evaluated, recognising that there is a relationship between the accuracy of recognition, the experience of the users and the acceptability of functional gestures.

The Freeform run mode offers limited editing, more editing functionality maybe useful. Also we have offered less behaviour in this mode than other tools, a more formal evaluation of what is appropriate in a run mode would be useful. Too much functionality may stifle creativity in the same way, as formal diagrams disrupt the design process.

The studies that both Bailey and Konstan (2003) and we have conducted have been small, discrete exercises. Larger studies with both students and practitioners would provide insights into the use of such tools throughout the software development life cycle.

Sketch tools rely on pen input. We are excited by the prospect of both better hardware and support for such tools in operating systems, which in turn will make it possible for informal interfaces to become the norm.

# Bibliography


Abowd, G. D., Atkeson, C. G., Feinstein, A., Hmelo, C., Kooper, R., Long, S., et al. (1996). Teaching and Learning as Multimedia Authoring: The Classroom 2000 Project, *Proceedings of ACM Multimedia 96*, Boston, MA, pp. 187-198.

Ainsworth, L. A., & Huddleston, C. (1999). *Interactive Whiteboards Are Bringing National Grid for Learning Resources under the Teachers Control for the Whole Class*. Retrieved 25/5/2000, 2000, from www.promethean.co.uk/stwilf.cs.htm

Alvarado, C., & Davis, R. (2001). Resolving Ambiguities to Create a Natural Computer-Based Sketching Environment, *Proceedings of IJCAI-01*, pp. 1365-1374.

Apperley, M., Dahlberg, B., Jefferies, A., Paine, L., Phillips, M., & Rogers, B. (2001). Lightweight Capture of Presentations for Review, *Proceedings of IHM-HCI*, Lille, pp. 41-42.

Apperley, M., & Duncan, A. (1994). Human-Computer Interface Design in the Software Lifecycle, *Proceedings of Software Engineering Conference*, Dunedin, pp. 60-64.

Apperley, M., Jansen, S., Jefferies, A., Masoodian, M., Paine, L., Rogers, B., et al. (2002). Llc Lecture Capture and Editing Tool for Online Course Delivery, *Proceedings of E-Learn, World Conference of E-Learning in Corporate, Government, Healthcare and Higher Education*, Montreal, pp. 1866-1869.

Bailey, B. P., & Konstan, J. A. (2003). Are Informal Tools Better? Comparing Demais, Pencil and Paper, and Authorware for Early Multimedia Design, *Proceedings of CHI 2003*, Ft Lauderdale, pp. 313-320.

Bailey, B. P., Konstan, J. A., & Carlis, J. V. (2001a). Demais: Designing Multimedia Applications with Interactive Storyboards, *Proceedings of ACM Multimedia*, pp. pp. 241-250.

Bailey, B. P., Konstan, J. A., & Carlis, J. V. (2001b). Supporting Multimedia Designers: Towards More Effective Design Tools, *Proceedings of Multimedia Modelling*, pp. pp. 267-286.

Barnes, D., Fincher, S., & Thompson, S. (1997). Introductory Problem Solving in Computer Science, *Proceedings of 5th Annual Conference on Teaching Computing*, Dublin, Ireland, pp. 36-39.

Bekker, M. M. (1993). Representational Issues Related to Communication in Design Teams, *Proceedings of Chi '93*, pp. 151-152.

Black, A. (1990). Visible Planning on Paper and on Screen: The Impact of Working Medium on Decision-Making by Novice Graphic Designers. *Behaviour and information technology, 9*(4), pp. 283-296.

Blum, B. (1992). *Software Engineering: A Holistic View*. New York: Oxford University Press.

Bly, S. A., & Minneman, S. L. (1990). Commune: A Shared Drawing Surface, *Proceedings of Conference on Office Information Systems*, pp. 184-191.

Bødker, S., Krogh, P., & Petersen, M. G. (2001). The Interactive Design Collaboratorium, *Proceedings of Interact '01*, Tokyo, pp. 51-58.

Booth, S. (1990). *Concepts of Programming: A Study into Learning to Program*, from http://ericae.net/ericdb/ED338227.htm

Cañas, J. J., Bajo, M. T., & Gonzalvo, P. (1994). Mental Models and Computer Programming. *International Journal of Human-Computer Studies, 40*, pp. 795-811.

Carroll, J. M. (1990). Infinite Detail and Emulation in an Ontologically Minimized Hci, *Proceedings of CHI 90*, pp. 321-327.

Carroll, J. M. (1996). Becoming Social: Expanding Scenario-Based Approaches to Hci. *Behaviour and information technology, 15*(4), pp. 266-275.

Carroll, J. M. (2000). Making Use: Scenarios and Scenario-Based Design, *Proceedings of Symposium on Designing Interactive Systems*, pp. 4.

Costabile, M. F. (2001). Usability in the Software Life Cycle. In *Handbook of Software Engineering and Knowledge Engineering* (Vol. 1, pp. 179-192): World Scientific Publishing.

Damm, C. H., Hansen, K. M., & Thomsen, M. (2000a). Tool Support for Cooperative Object-Oriented Design: Gesture Based Modelling on and Electronic Whiteboard, *Proceedings of Chi 2000*, pp. 518-525.

Damm, C. H., Hansen, K. M., Thomsen, M., & Tyrsted, M. (2000b). Supporting Several Levels of Restriction in the Uml, *Proceedings of UML 2000*, York, UK, pp. 518-525.

Denzin, N. K. (1988). Triangulation. In K. J.P (Ed.), *Educational Research, Methodology, and Measurement: An International Handbook* (pp. 511-513). Sydney: Pergamon Press.

Détienne, F. (1990). Expert Programming Knowledge: A Schema Based Approach. In J.-M. Hoc, T. R. G. Green, R. Samurcay & D. J. Gilmore (Eds.), *Psychology of Programming* (pp. 205-222). London: Academic Press.

Do, E. Y. L., & Gross, M. (2001). Thinking with Diagrams in Architectural Design. *Artificial Intelligence Review*(15), pp. 135-149.

Elrod, S., Bruce, R., Gold, R., Goldberg, D., Halasz, F., Janssen, W., et al. (1992). Liveboard: A Large Interactive Display Supporting Group Meetings, Presentations and Remote Collaboration. *CHI '92*, pp. 599-607.

Fincher, S. (1999). What Are We Doing When We Teach Programming?, *Proceedings of 29th ASEE/IEEE Frontiers in Education*, San Juan Puerto Rico, pp.

Goel, V. (1995). *Sketches of Thought*. Cambridge, Massachusetts: The MIT Press.

Goldschmidt, G. (1991). The Dialectics of Sketching. *Creative Research Journal, 4*(2), pp. 123-143.

Goldschmidt, G. (1992). Serial Sketching: Visual Problem Solving in Design, the Backtalk of Self-Generated Sketches. *Cybernetics and Systems, 23*, pp. 191-219.

Goldschmidt, G. (1999). The Backtalk of Self-Generated Sketches. In J. S. Gero & B. Tversky (Eds.), *Visual and Spatial Reasoning in Design* (pp. 163-184). Sydney: Key Centre, University of Sydney.

Green, T. R. G. (1990). Programming Languages as Information Structures. In J.-M. Hoc (Ed.), *Psychology of Programming*. London: Academic Press.

Greenaway, R. (2003). *Experiential Learning Cycles*. Retrieved 24/6/03, 2003, from http://reviewing.co.uk/research/learning.cycles.htm

Gross, M. (1994). Recognizing and Interpreting Diagrams in Design, *Proceedings of AVI 94*, Bari, Italy, pp. 88-94.

Gross, M. (1998). The Proverbial Back of an Envelope. *IEEE Intelligent Systems,* 10-13.

Gross, M., & Do, E. Y. L. (1996). Ambiguous Intentions: A Paper-Like Interface for Creative Design, *Proceedings of UIST '96*, Seattle Washington, pp. 183-192.

Hammond, T., & Davis, R. (2003). Ladder: A Language to Describe Drawing, Display, and Editing in Sketch Recognition, *Proceedings of IJCAI*, pp. 12-19.

Hearst, M. A. (1998). Sketching Intelligent Systems. *IEEE Intelligent Systems, May/June*(13 (3)), pp. 10-19.

Insight Development Corp. (2000). *Squiggle*, 2002, from http://www.residential.com/squiggle.html

Johnson, P. (1992). *Human Computer Interaction: Psychology, Task Analysis and Software Engineering*. London: McGraw-Hill.

Kay, J., Barg, M., Fekete, A., Greening, T., Hollands, O., Kingston, J. H., et al. (2000). Problem-Based Learning for Foundation Computer Science Courses. *Computer Science Education, 10*(2), pp. 109-128.

Knutilla, A. J., Steves, M. P., & Allen, R. H. (2001). Workshop on Evaluating Collaborative Enterprises - Workshop Report, *Proceedings of IEEE 9th International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WET ICE'00)*, Gaithersburg, Maryland, pp. 79-85.

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. (1995). Interactive Sketching for User Interface Design, *Proceedings of Chi '95 Mosaic of Creativity, Doctoral Consortium*, ACM, pp. 63-64.

Landay, J. (1996). *Interactive Sketching for the Early Design Stages of User Interface Design*, Thesis, Carnegie Mellon University, Pittsburgh, PA.

Landay, J. (1998). Informal User Interfaces for Natural Human-Computer Interaction. *IEEE Intelligent Systems,* 14-16.

Landay, J. (1999). Using Note-Taking Appliances for Student to Student Collaboration, *Proceedings of 29th ASEE/IEEE Frontiers in education conference*, San Juan, Puerto Rico, pp. 15-20.

Landay, J., & Myers, B. (1995). Interactive Sketching for the Early Stages of User Interface Design, *Proceedings of Chi '95 Mosaic of Creativity*, ACM, pp. 43-50.

Landay, J., & Myers, B. (1996). Sketching Storyboards to Illustrate Interface Behaviors, *Proceedings of CHI '96*, Vancouver, BC Canada, pp. 193-194.

Landay, J., & Myers, B. (2001). Sketching Interfaces: Toward More Human Interface Design. *Computer, 34*(3), pp. 56-64.

Lin, J., Newman, M. W., Hong, J. I., & Landay, J. A. (2000). Denim: Finding a Tighter Fit between Tools and Practice for Web Design, *Proceedings of Chi 2000*, pp. 510-517.

Linn, M. C. (1985). The Cognitive Consequences of Programming Instruction in Classrooms. *Educational Researcher, 14*(5), pp. 14-29.

Linn, M. C., & Clancy, M. J. (1992). Case for Case Studies of Programming Problems. *Communications of the ACM .Mar 1992, 35*(3), pp. 121-132.

Long, A. C. (1998). Improving Gestures and Interaction Techniques for Pen-Based User Interfaces, *Proceedings of CHI 98*, Los Angles, CA, pp. 58-59.

Long, A. C., Landay, J. A., & Rowe, L. A. (1999). Implications for a Gesture Design Tool, *Proceedings of CHI 99*, Pittsburgh PA, pp. 40-47.

Long, A. C., Landay, J. A., Rowe, L. A., & Michiels, J. (2000). Visual Similarity of Pen Gestures, *Proceedings of CHI 2000*, The Hague, Amsterdam, pp. 360-367.

Maheshwari, P. (1997). Improving the Learning Environment in First-Year Programming: Integrating Lectures, Tutorials and Laboratories. *Journal of Computers in Mathematics and Science Teaching, 16*(1), pp. 111-131.

Marsden, G. (1997). *Designing Graphical Interface Programming Languages for End User Programming*, Thesis, University of Stirling, Scotland.

Marsden, G., & Thimbleby, H. (1998). Holistic Programming Environments. *South African Computer Journal, 21*, pp. 237-241.

Mayer, R. E. (1976). Some Conditions of Meaningful Learning for Computer Programming: Advance Organizers and Subject Control of Frame Order. *Journal of Educational Psychology, 67*, pp. 725-734.

McConnell, S. (1996). *Rapid Development : Taming Wild Software Schedules*. Redmond: Microsoft Press.

McQueen, M., & Mann, S. (2000). A Language Model Based Optical Character Recogniser (Ocr) for Reading Incidental Text, *Proceedings of National Advisory Committee on Computing Qualifications*, Wellington, pp. 207-212.

Milbrandt, G. (1995). Using Problem Solving to Teach a Programming Language. *Learning and Leading with Technology, 23*(2), pp. 27-31.

Mimio. (1999). *Mimio*, from http://www.virtualink.com

Minneman, S. L., & Bly, S. A. (1991). Managing a Trois: A Study of Multi-User Drawing Tool in Distributed Design Work, *Proceedings of Human factors in computing systems*, pp. 217-224.

Moran, T. P., Chiu, P., & van Melle, W. (1997). Pen-Based Interaction Techniques for Organizing Material on an Electronic Whiteboard, *Proceedings of 10th Annual Symposium on User Interface Software and Technology*, Banff, Canada, pp. 45-54.

Moran, T. P., van Melle, W., & Chiu, P. (1998). Spatial Interpretation of Domain Objects Integrated into a Freeform Electronic Whiteboard, *Proceedings of UIST '98*, San Francisco, CA, pp. 175-184.

Myers, B. (1994). Challenges of Hci. *Interactions, 1*(1), pp. 73-83.

Myers, B., & Rosson, M. B. (1992). Survey on User Interface Programming, *Proceedings of Chi '92*, pp. 195-202.

Mynatt, E. D., Igarashi, T., Edwards, W. K., & LaMarca, A. (1999). Flatland: New Dimensions in Office Whiteboards, *Proceedings of CHI 99*, Pittsburgh, pp. 346-353.

Navarro, R., & Cañas, J. J. (2001). Are Visual Programming Languages Better? The Role of Imagery in Program Comprehension. *International Journal of Human-Computer Studies, 54*, pp. 799-829.

Navarro, R., Cañas, J. J., & Bajo, M. T. (1996). Pictorial Aids in Computer Use, *Proceedings of 8th European Conference on Cognitive Ergonomics*, Granada, pp. 77-82.

Navarro, R., Cañas, J. J., & Green, T. R. G. (1997). Mental Representation and Imagery in Program Understanding: C and Spreadsheets, *Proceedings of Empirical Studies of Programmers 7 Conference*, Virginia, USA, pp. 234-235.

Navarro, R., & Green, T. R. G. (1995). Programming Plans, Imagery and Visual Programming, *Proceedings of INTERACT-95*, London, pp. 139-144.

Newell, A., & Simon, H. A. (1972). *Human Problem Solving*. Englewood Cliffs, NJ: Prentice-Hall.

Newman, M. W., & Landay, J. A. (2000). Sitemaps, Storyboards and Specifications, *Proceedings of Designing interactive systems: processes, practices, methods and techniques*, Brooklyn, NY, pp. 263-274.

Newman, M. W., Lin, J., Hong, J. I., & Landay, J. A. (2003). Denim: An Informal Web Site Design Tool Inspired by Observations of Practice. *Human-Computer Interaction, 18*(3), pp. 259-324.

Nielsen, J. (1994). *Usability Engineering*. San Francisco: Morgan Kaufmann.

Papert, S. (1980). *Mindstorms: Children, Computers and Powerful Ideas*. New York: Basic Books.

Pedersen, E. R., McCall, K., Moran, T. P., & Halasz, F. G. (1993). Tivoli: An Electronic Whiteboard for Informal Workgroup Meetings, *Proceedings of CHI '93*, pp. 391-398.

Pennington, N. (1987a). Comprehension Strategies in Programming, *Proceedings of Empirical studies of programmers: second workshop*, Norwood, NY, pp. 100-112.

Pennington, N. (1987b). Stimulus Structures and Mental Representation in Expert Comprehension of Computer Programs. *Cognitive Psychology, 19*, pp. 295-341.

Pennington, N., & Grabowski, B. (1990). The Task of Programming. In H. J.M., T. R. G. Green, R. Samurcay & D. J. Gilmore (Eds.), *Psychology of Programming* (pp. 45-62). London: Academic Press.

Petre, M., & Blackwell, A. F. (1999). Mental Imagery in Program Design and Visual Programming. *International Journal of Human-Computer Studies, 51*, pp. 7-30.

Pinto-Albuquerque, M., Fonseca, M., & Jorge, J. (2000). Visual Languages for Sketching Documents, *Proceedings of 2000 IEEE International Symposium on Visual Languages (VL'00)*, pp. 225-231.

Plimmer, B. E., & Apperley, M. (2001a). Freeform: Informal Form Design on a Large Interactive Display Surface, *Proceedings of Symposium on computer human interaction*, Palmerston North, New Zealand, pp. 81-83.

Plimmer, B. E., & Apperley, M. (2001b). From Sketch to Form on a Large Interactive Display Surface, *Proceedings of National Advisory Committee on Computing Qualifications*, Napier, pp. 371-374.

Plimmer, B. E., & Apperley, M. (2002a). Computer-Aided Sketching to Capture Preliminary Design, *Proceedings of Australian User Interface Conference*, Melbourne, pp. 9-12.

Plimmer, B. E., & Apperley, M. (2002b). Freeform: An Informal Environment for Interface Prototyping, *Proceedings of CHINZ*, Hamilton, pp. 11-12.

Plimmer, B. E., & Apperley, M. (2003a). Evaluating a Sketch Environment for Novice Programmers, *SIGCHI* (pp. 1018-1019). Ft Lauderdale: ACM.

Plimmer, B. E., & Apperley, M. (2003b). Freeform: A Tool for Sketching Form Designs, *BHCI* (Vol. 2, pp. 183-186). Bath.

Plimmer, B. E., & Apperley, M. (2003c). Software for Students to Sketch Interface Designs, *Interact* (pp. 73-80). Zurich.

Plimmer, B. E., & Apperley, M. (2004). Interacting with Sketched Interface Designs, *SIGCHI* (Vol. 2). Vienna: ACM.

Prasad, C., & Fielden, K. (2002). Introducing Programming: A Balanced Approach, *Proceedings of National Advisory Committee on Computing Qualifications*, Hamilton, pp. 101-106.

Rettig, M. (1994). Prototyping for Tiny Fingers. *Communications of the ACM, 37*(4), pp. 21-27.

Rist, R. S. (1995). Program Structure and Design. *cognitive Science, 19*, pp. 501-562.

Rist, R. S. (1996). Teaching Eiffel as a First Language. *Journal of object-oriented programming, 9*, pp. 30-41.

Robins, A., Rountree, J., & Rountree, N. (2003). Learning and Teaching Programming: A Review and Discussion. *Computer Science Education, 13*(2), pp. 137-172.

Rubine, D. (1991). Specifying Gestures by Example, *Proceedings of Proceedings of Siggraph '91*, pp. 329-337.

Rudd, J., Stern, K., & Isensee, S. (1996). Low Vs High-Fidelity Prototyping Debate. *Interactions, 3*(1), pp. 76-85.

Shneiderman, B. (1980). *Software Psychology*. Cambridge, Mass.: Winthrop Publishers.

Smartboard. (2000). *Smartboard*, from http://www.smarttech.com

Soloway, E. (1984). Empirical Studies of Programming Knowledge. *Transactions on Software Engineering, 10*(5), pp. 595 -609.

Stahovich, T. F. (1997). Interpreting the Engineer's Sketch: A Picture Is Worth a Thousand Constraints, *Proceedings of AAAI Symposium on Reasoning with diagrammatic Representations II*, Cambridge, Mass., pp. 31-38.

Stahovich, T. F. (1998). The Engineering Sketch. *IEEE Intelligent Systems*(13 (3)), pp. 17-19.

Stahovich, T. F., Davis, R., & Shrobe, H. (1995). Turning Sketches into Working Geometry, *Proceedings of ASME Design Theory and Methodology*, pp. 603-611.

Stahovich, T. F., Davis, R., & Shrobe, H. (1996). Generating Multiple New Designs from a Sketch. *AAAI-96*, pp. 1022-1029.

Strothotte, T., Preim, B., Raab, A., Schumann, J., & Forsey., D. R. (1994). How to Render Frames and Influence People, *Proceedings of Eurographics '94*, Oslo, Norway, pp. 455-466.

Sutherland, I. E. (1963). Sketchpad: A Man-Machine Graphical Communication System, *Proceedings of Spring joint computer conference: American Federation Information Processing Societies*, Montvale, New Jersey, pp. 329-346.

Trinder, M. (1999). The Computer's Role in Sketch Design: A Transparent Sketching Medium,, *Proceedings of Computers and Building, CAAD futures 99*, Atlanta, pp. 227-244.

Tversky, B. (1999). What Does Drawing Reveal About Thinking, *Proceedings of Visual and spatial reasoning in design*, Cambridge, Mass., pp. 75-81.

Wagner, A. (1990). Prototyping: A Day in the Life of an Interface Designer. In B. Laurel (Ed.), *The Art of Human-Computer Interface Design* (pp. 79-84). Reading MA: Addison-Wesley.

Wong, Y. Y. (1992). Rough and Ready Prototypes: Lessons from Graphic Design, *Proceedings of Human Factors in Computing Systems CHI '92*, Monterey, pp. 83-84.

Wright, T., & Cockburn, A. (2002). Mulspren: A Multiple Language Simulation Programming Environment, *Proceedings of HCC 02*, Arlington, Virginia, pp. 101-103.

# Appendix 1

# First Usability Study

**Participant Guide**


The University of Waikato

# Usability Laboratory

## Visual Basic Sketch Environment

## Usability Study

**Participant Workbook**

**Session id:** **Date and time:**

**Participant name:** **Participant id:**

Updated 12th September 2001

**The University of Waikato Department of Computer Science**

Research Consent Form

*This consent form, a copy of which has been given to you, is only part of the process of informed consent. It should give you the basic idea of what the research is about and what your participation will involve. If you would like more detail about something mentioned here, or information not included here, please ask. Please take the time to read this form carefully and to understand any accompanying information.*

**Research Project Title**

**Large Interactive Display Sketch Interface for Program Design.**

**Researcher**

Beryl Plimmer

**Experiment Purpose**

The purpose of this experiment is to observe and record the ways that programming students design computer forms using both traditional tools and a large interactive display sketch interface.

**Participant Recruitment and Selection**

Undergraduate and graduate students from the University of Waikato and Manukau Institute of Technology are being recruited for this experiment. At the time of the experiment the participants cannot be members of a course that Beryl Plimmer is teaching.

**Procedure**

This session will require about sixty minutes of your time. You will be asked to sketch a form design for a simple programming problem. None of the tasks is a test – my objective is to find out how you approach the tasks and test the usability of the computer based sketch environment.

**Data Collection**

An observer will take notes as they watch you work. The session will also be recorded on video for further analysis. These videotapes will be destroyed at the completion of the project.

**Confidentiality**

Confidentiality and participant anonymity will be strictly maintained. All information gathered will be used for statistical analysis only and no names or other identifying characteristics will be stated in the final or any other reports.

**Likelihood of Discomfort**

There is no likelihood of discomfort or risk associated with participation.

**Researcher**

Beryl Plimmer is working on her doctorate in the Computer Science Department at the University of Waikato. This study will contribute to her

research on large interactive sketch interfaces. Her supervisor is Professor Mark Apperley.

Beryl can be contacted at Manukau Institute of Technology, Information Systems Department. Her phone number is (09) 2746009 extension 7453 and her email address is beryl.plimmer@manukau.ac.nz, or bep2@cs.waikato.ac.nz

**Finding out about Results**

The Participants can find out the results of the study by contacting the researcher after December 1 2003

**Agreement**

Your signature on this form indicates that you have understood to your satisfaction the information regarding participation in the research project and agree to participate as a participant. In no way does this waive you legal rights nor release the investigators, sponsors, or involved institutions from their legal and professional responsibilities. You are free to withdraw from the study at any time without penalty. Your continued participation should be as informed as your initial consent, so you should feel free to ask for clarification or new information throughout your participation. If you have further questions concerning matters related to this research, please contact the researcher.

_____ _____

Participant Date

_____ _____

Investigator/Witness Date

A copy of this consent form has been given to you to keep for your records and reference.

## 1. Introduction

Thank you for participating in our study. The main purpose of the study is to assess the usability of the software when participant uses the sketch environment to complete a small form design task. The study will identify any difficulties that the participants may have in using the environment (in particular.) The study will, observe the participants as they actually use the environment, and ascertain the participant's interaction experiences. Further, the study will endeavour to learn from the participants the ease or difficulty they had with the environment and how useful they found the product in supporting form design.

Firstly we will spend a few minutes familiarising you with the equipment and software. Once we have done this we will create a small form together in the sketch environment. We will then give you a scenario and ask you to design a form for that scenario. We will then discuss with you any issues or difficulties you had with the system, as well as anything you found easy to use.

*Note*: You may leave this programme at any time that you wish.

a) LIDS (interactive whiteboard)

b) LIDS Training

The researcher will now go through a brief safety and training session with you.

Please ask any questions that come to mind.

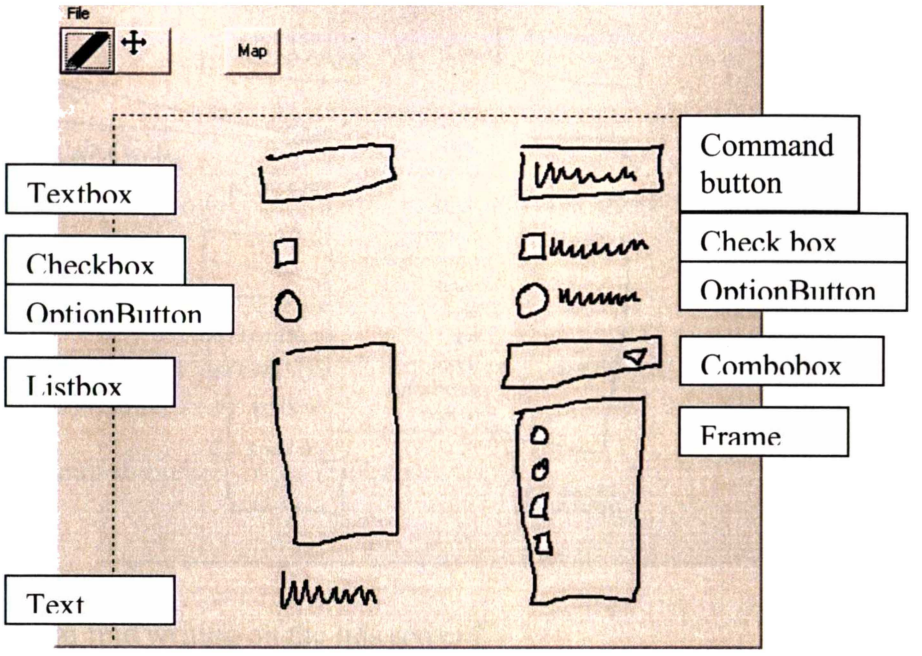
c) Sketch Environment Training

The sketch environment you are about to use works by recognizing the shapes of the things that you draw. Each shape must be drawn as a single pen stroke. If you lift your pen off the surface and place it back down this will make another shape.

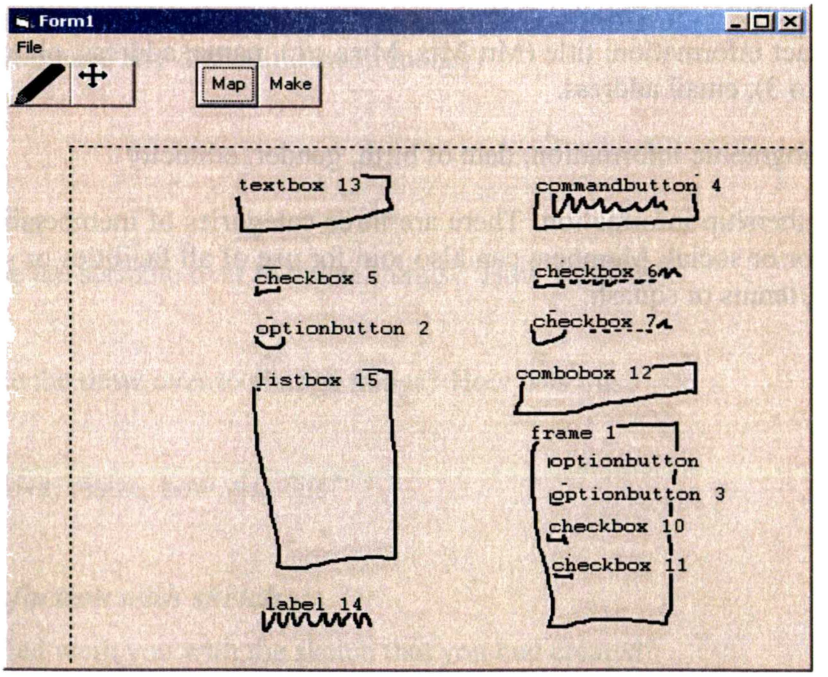
The software operates in four different modes

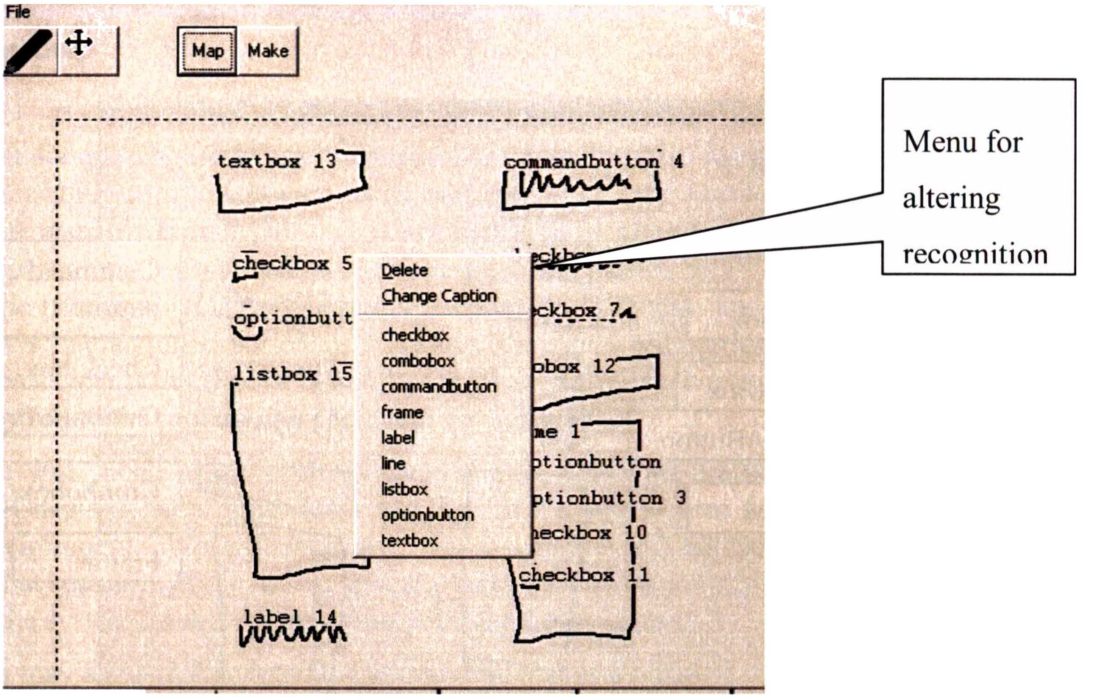
- Drawing –used for drawing control strokes
- Writing – for words
- Editing – moving and resizing
- Map – the recogniser engine maps your drawing to VB controls, you may then alter any of these mappings
- Make – the VB form is created from the current sketch – this is only available after you have mapped

The environment recognises a number of basic shapes and can combine two shapes to make a control




A mapped form shows the names of the controls in labels, click the labels to show a menu of available controls






d) Design Task

Please design a form for the Waikato sports club to use for member details. The form should show the following:

- contact information: title (Mr, Mrs, Miss etc), name, address, phone number (up to 3), email address.
- Demographic information: date of birth, gender, ethnicity
- Membership information: There are three categories of membership, junior, senior or social. Members can also join for use of all facilities or specify for golf, tennis or squash.

**e) Post-Task Interview Notes**

We would like to understand how you found the experience of using the sketch interface

### *1. Basic drawing*

How did you find drawing on the lids screen (pen/ screen)?

Did you have any problems making the shapes you intended?

Was it difficult to create single stroke shapes?

### *2. Basic writing*

How did you find writing on the lids screen?

### *3. Writing / Drawing Mode*

Was it clear to you which mode you were in? Was it difficult to change modes? Did you have any difficulties with modes?

### *4. Editing*

Did you use the scribble over to delete things? How was this?

Did you use the draw over to change things? How was this?

Did you move, resize, how was this?

### *5. Satisfaction with sketch*

How satisfied were you with the sketch that you had created?

### *6. Mapping*

Did you alter any of the recognition after mapping? How did you find this?

### *7. Make*

Was the form as you expected it?

How satisfied were you with the form?

If there were three things you would like to change what would they be

- 
- 
- 

What features you would like to see on future versions.

- 
- 
- 

Thank you

# 2. Summary of Post Task Questionnaires

Basic drawing

How did you find drawing on the lids screen (pen/ screen)?

1. *ok, quite comfortable, smooth*
2. *Easier than mouse*
3. *pen sound annoying*
4. *Easier than mouse (x3) – more direct*
5. *Pretty responsive*
6. *At first slow, I thought it might break*

Did you have any problems making the shapes you intended?

1. *no, but frame – too small became a command button*
2. *not really, eraser not always available*
3. *option buttons (made checkboxes)*
4. *no*
5. *no – one circle became checkbox – not a problem*
6. *no*

Was it difficult to create single stroke shapes?

1. *ok*
2. *no problems*
3. *fine*
4. *fine as he had been told – would not have guessed*
5. *no – maybe big frame more difficult*
6. *no, it was all right*

### *1. Basic writing*

How did you find writing on the lids screen?

1. *hard, the physical pen was uncomfortable and nib movement annoying*
2. *not difficult would like recognition*
3. *irrelevant as not recognised – pen too big, didn't like writing*
4. *anything ok as not interpreted, nib up/down yuk*
5. *sweet*
6. *harder, but fun*

### *2. Writing / Drawing Mode*

Was it clear to you which mode you were in? Was it difficult to change modes? Did you have any difficulties with modes?

1. *only one problem then fairly easy*
2. *ok*
3. *comfortable*
4. *One mistake – different colours helps*

5. *knew when wrong, could erase and redo*
6. *clear, no problem to change*

### 3. *Editing*

Did you use the scribble over to delete things? How was this?

1. *no*
2. *program crash –bug found after test 3*
3. *didn't work (not sure why)*
4. *ok*
5. *once happened by accident – intended command button*
6. *good, easy, fast*

Did you use the draw over to change things? How was this?

1. *no*
2. *n/a*
3. *didn't work (not close enough match)*
4. *n/a*
5. *n/a*
6. *n/a*

Did you move, resize, how was this?

1. *trouble picking up small items (changed after test3)*
2. *easier in vb as you don't have to grab middle handle in vb to move*
3. *finding handles a bit hard, same with mouse, probably end up quicker*
4. *good*
5. *fine like grouping*
6. *ok moving, didn't need resize*

### 4. *Satisfaction with sketch*

How satisfied were you with the sketch that you had created?

1. *good got everything needed*
2. *ok*
3. *ok yes*
4. *depends on what's acceptable to others – quicker than form – gives less time to think*
5. *fine could change layout*
6. *happy with it*

### 5. *Mapping*

Did you alter any of the recognition after mapping? How did you find this?

1. *ok*
2. *good*
3. *easy*
4. *just like ordinary*
5. *simple*

6. *easy*

### 6. *Make*

Was the form as you expected it?

1. *more or less, messed up combos*
2. *ok too many labels*
3. *ok*
4. *wasn't sure what to expect*
5. *yeh*
6. *as expected but should line up*

How satisfied were you with the form?

1. *ok,*
2. *almost would like ink colours to go to form*
3. *ok*
4. *would like it to be tidy /aligned*
5. *exactly what was drawn*
6. *quite happy*

If there were three things you would like to change what would they be

1.
    a. *ink thickness*
    b. *pen nib smaller*
2.
    a. *Eraser – fix bug*
    b. *Difficult to resize/move*
3.
    a. *Write outside border – ran out of space (told him he could)*
    b. *Icons more different for write/draw suggested pen and ruler for draw*
    c. *Cleaner screen (? Check video)*
4.
    a. *handles that look like handles (bar with a D)*
5.
    a. *should be lined up*
6.
    a. *lining up*

What features you would like to see on future versions.

1.

   a. *handwriting recognition*
   b. *add things to lists in sketch mode (eg Mr, Mrs, Miss)*

2.

   a. *handwriting recognition*
   b. *command buttons to be a different glyph to textboxes (suggested oval – ok)*

3.

   a. *handwriting recognition/ voice recognition*
   b. *the textboxes on form not to overlap (tidy form)*
   c. *default text can be entered in text boxes*

4.

   a. *handwriting recognition*
   b. *colours in end product for little kids (?)*

5.

   a. *handwriting recognition*
   b. *code for things like exit*

6.

   a. *nothing*

## *7. Known Errors*

*An error occurs when removing controls from form before placing new ones on if the form had been 'run' probably something to do with what is currently selected*

User Requirements

*Character recognition (5)*

*Undo/redo (3)*

*Form tidy and align (4)*

# Appendix 2

## Second Usability Study

---

# Participant Guide

# Visual Basic Sketch Environment

# Usability Study

## 1.01 Participant Workbook

**Session id:** **Date and time:**

**Participant name:** **Participant id:**

# Research Consent Form

*This consent form, a copy of which has been given to you, is only part of the process of informed consent. It should give you the basic idea of what the research is about and what your participation will involve. If you would like more detail about something mentioned here, or information not included here, please ask. Please take the time to read this form carefully and to understand any accompanying information.*

## Research Project Title

**Large Interactive Display Sketch Interface for Program Design**.

## Researcher

Beryl Plimmer

## Experiment Purpose

The purpose of this experiment is to observe and record the ways that programming students design computer forms using both traditional tools and a large interactive display sketch interface.

## 3. Participant Recruitment and Selection

Undergraduate and graduate students from the University of Waikato and Manukau Institute of Technology are being recruited for this experiment. At the time of the experiment the participants cannot be members of a course that Beryl Plimmer is teaching.

## Procedure

This session will require about sixty minutes of your time. You will be asked to sketch a form designs for a simple programming problem. None of the tasks is a test - my objective is to find out how you approach the tasks and test the usability of the computer based sketch environment.

## Data Collection

An observer will take notes as they watch you work. The session will also be recorded on video for further analysis. These videotapes will be destroyed at the completion of the project.

## Confidentiality

Confidentiality and participant anonymity will be strictly maintained. All information gathered will be used for statistical analysis only and no names or other identifying characteristics will be stated in the final or any other reports.

## Likelihood of Discomfort

There is no likelihood of discomfort or risk associated with participation.

## Researcher

Beryl Plimmer is working on her doctorate in the Computer Science Department at the University of Waikato. This study will contribute to her research on large interactive sketch interfaces. Her supervisor is Professor Mark Apperley.

Beryl can be contacted at Manukau Institute of Technology, Computing and IT Department. Her phone number is (09) 9688765 extension 7453 and her email address is beryl.plimmer@manukau.ac.nz, or bep2@cs.waikato.ac.nz

## Finding out about Results

The Participants can find out the results of the study by contacting the researcher after December 1 2003

## Agreement

Your signature on this form indicates that you have understood to your satisfaction the information regarding participation in the research project and agree to participate as a participant. In no way does this waive you legal rights nor release the investigators, sponsors, or involved institutions from their legal and professional responsibilities. You are free to withdraw from the study at any time without penalty. Your continued participation should be as informed as your initial consent, so you should feel free to ask for clarification or new information throughout your participation. If you have further questions concerning matters related to this research, please contact the researcher.

_____ _____

Participant Date

_____ _____

Investigator/Witness Date

A copy of this consent form has been given to you to keep for your records and reference.

# 1. Introduction

Thank you for participating in our study. The main purpose of the study is to assess the usability of the software when participant uses the sketch environment to complete a small form design task. The study will identify any difficulties that the participants may have in using the environment (in particular.) The study will, observe the participants as they actually use the environment, and ascertain the participant's interaction experiences. Further, the study will endeavour to learn from the participants the ease or difficulty they had with the environment and how useful they found the product in supporting form design.

Firstly we will spend a few minutes familiarising you with the equipment and software. Once we have done this we will create a small form together in the sketch environment. We will then give you a scenario and ask you to design a form for that scenario. We will then discuss with you any issues or difficulties you had with the system, as well as anything you found easy to use.

*Note*: You may leave this programme at any time that you wish.

# a. LIDS (interactive whiteboard)

## b) LIDS Training

The researcher will now go through a brief safety and training session with you.

Please ask any questions that come to mind.

## c) Sketch Environment Training

The sketch environment you are about to use works by recognizing the shapes of the things that you draw. Each shape must be drawn as a single pen stroke. If you lift your pen off the surface and place it back down this will make another shape.

The software operates in five different modes


- Drawing –used for drawing control strokes
- Writing – for words
- Editing – moving and resizing
- Run – used to work through use case scenarios
- Map – the recogniser engine maps your drawing to VB controls, you may then alter any of

these mappings

- Make – the VB form is created from the current sketch – this is only available after you have mapped

The environment recognises a number of basic shapes and can combine two shapes to make a control

Text box

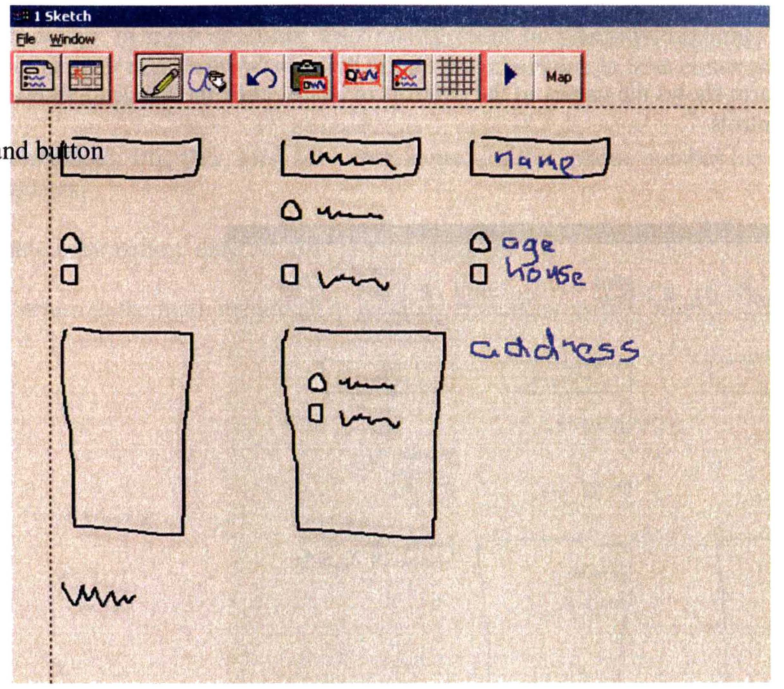
command button

option button

check box

list box

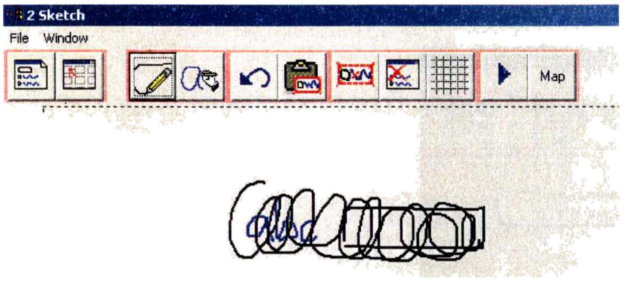
label



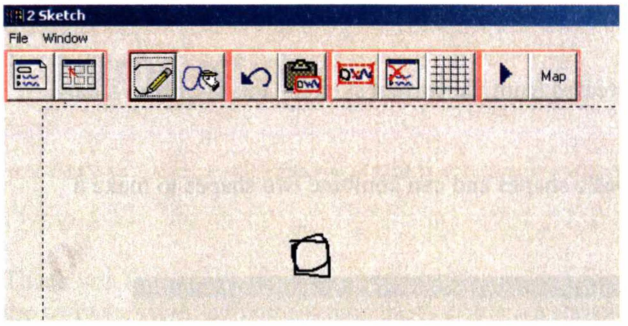

frame

Editing your sketch

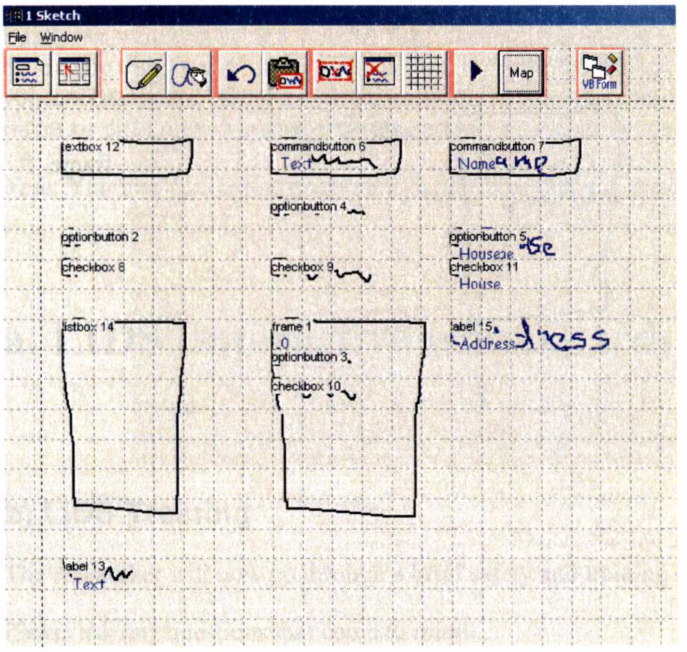
You can delete ink by scribbling over it with a delete gesture

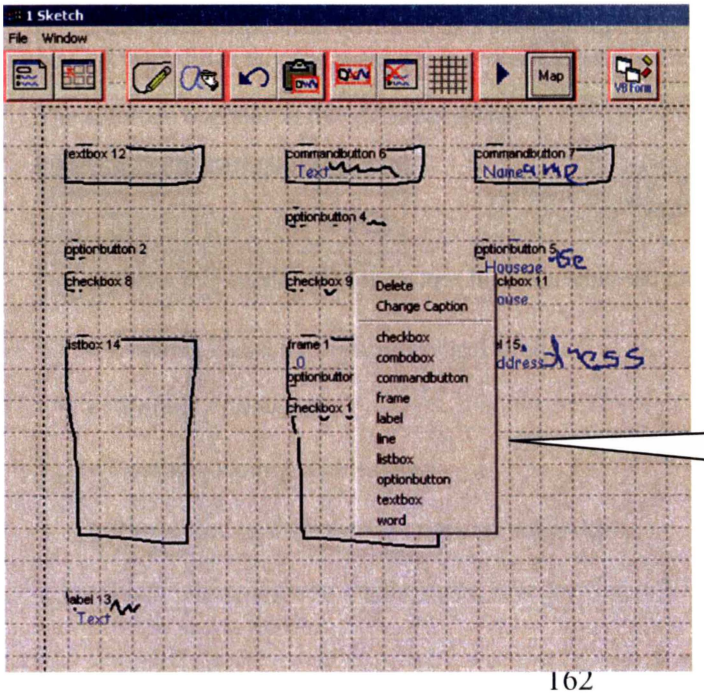



Drawing over a shape with another about the same size will replace the first shape (for example to change an option button to a text box)




A mapped form shows the names of the controls in labels, click the labels to show a menu of available controls




Menu for altering recognition

## d) Design Task

Please design two forms for the Manukau sports. The first form is a menu to take the users to forms for member details, member accounts, asset management, and facilities bookings. The second form is the member details form (you will not do the others due to time constraints. The form should show the following:

- Contact information: title (Mr, Mrs, Miss etc), name, address, phone number (up to 3), and email address.
- Demographic information: date of birth, gender, ethnicity
- Buttons to return to the main menu and to go to the member accounts form

## e) Post-Task Interview Notes

We would like to understand how you found the experience of using the sketch interface

**Basic drawing**

How did you find drawing on the lids screen (pen/ screen)?

Did you have any problems making the shapes you intended?

Was it difficult to create single stroke shapes?

**Basic writing**

How did you find writing on the lids screen?

**Writing / Drawing Mode**

Was it clear to you which mode you were in? Was it difficult to change modes? Did you have any difficulties with modes?

**Editing**

Did you use the scribble over to delete things? How was this?

Did you use the draw over to change things? How was this?

Did you move, resize, how was this?

**Satisfaction with sketch**

How satisfied were you with the sketch that you had created?

**Run mode**

Did you find run mode useful? Did you decide to change anything on your sketch after doing a 'run through?'

**Mapping**

Did you alter any of the recognition after mapping? How did you find this?

**Make**

Was the form as you expected it?

How satisfied were you with the form?

If there were three things you would like to change what would they be

- 
- 
- 

What features you would like to see on future versions.

- 
- 

Thank you

# 1. Summary of Post Task Questionnaires

### *Basic drawing*

How did you find drawing on the lids screen (pen/ screen)?

7. *pen fine, but distracted by drawing - ravi*
8. *initially hard but with a bit of practice ok, needed to recalibrate pen- angratal*
9. *exciting -arati*
10. *took a bit of getting used to – realising its not a mouse – then ok – phillip*
11. *alright, responsive - nath*
12. *easy to use – saurab*

Did you have any problems making the shapes you intended?

7. *didn't*
8. *not really*
9. *not exactly what I wanted sometimes had to rub out*
10. *absolutely ok*
11. *no*
12. *no*

Was it difficult to create single stroke shapes?

7. *no*
8. *a little difficult to start*
9. *no*
10. *initially a bit of a problem, then ok*
11. *some letters*
12. *no*

### *Basic writing*

How did you find writing on the lids screen?

7. *problems – difficult*
8. *fun, exciting!*
9. *alright, but letters not always joined together*
10. *initially uncomfortable, different from paper – still not used to it*
11. *easy*
12. *not comfortable to start*

### *1. Writing / Drawing Mode*

Was it clear to you which mode you were in? Was it difficult to change modes? Did you have any difficulties with modes?

7. *not really*
8. *mostly drawing – ok*

9. *bit confusing to start – get used to it*
10. *not clear to start then saw icons and clicked*
11. *yes because of colours*
12. *yes*

### *2. Editing*

Did you use the scribble over to delete things? How was this?

7. *hard*
8. *exciting*
9. *yes, cool*
10. *at first didn't cover whole text then ok*
11. *no – used undo*
12. *not used – used undo*

Did you use the draw over to change things? How was this?

7. *no*
8. *n/a*
9. *n/a*
10. *happened by accident*
11. *no*
12. *n/a*

Did you move, resize, how was this?

7. *hard*
8. *difficult*
9. *not difficult*
10. *ok when understood*
11. *ok except need help with multi- select*
12. *no*

### *Satisfaction with sketch*

How satisfied were you with the sketch that you had created?

7. *easy to design, but hard to make nice*
8. *very satisfied*
9. *enjoyed*
10. *looked messy – too messy to show to client/teacher*
11. *too big*
12. *good for a start, how would it be for a big project?*

### *Run mode*

1. *ok, didn't use much*
2. *cool*
3. *yes, no changes*
4. *ok*

5. *yes, can do before forms*
6. *alright*

### *3. Mapping*

Did you alter any of the recognition after mapping? How did you find this?

7. *ok – deleting and clearing easy*
8. *useful*
9. *helpful*
10. *able to get it right, easy*
11. *easy*
12. *alright*

### *4. Make*

Was the form as you expected it?

7. *– ran out of time*
8. *took a while*
9. *yes*
10. *ok – happy – tidy enough*
11. *yes*
12. *a bit of rubbish on screen I had to get rid of*

How satisfied were you with the form?

7. *–*
8. *good*
9. *yes – good*
10. *very good*
11. *ok*
12. *would like more naming*

If there were three things you would like to change what would they be

7.
    a. *menus*
8.
    a. *Edit mode, maybe colour with pen-down for edit and select*
    b. *Eraser*
9.
    a. *modes a bit confusing*
    b. *make draw and write the same*
    c. *better recognition*
10.
    a. *a tidier sketch*
    b. *icons easier to recognise*
11.
    a. *create code for links*

b. *targets for selects bigger*

c. *tool tips on icons*

*12.*

a. *name controls as go*

What features you would like to see on future versions.

*7.*

c. *join words side by side*

*8.*

d. *change size/colour of ink*

*9.*

c. *add words to icons*

*10.*

c. *code generation for links*

d. *map and make all forms together*

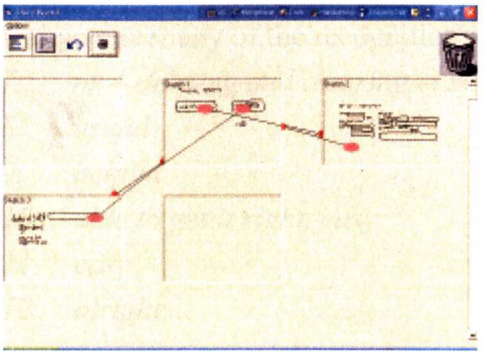
*11.*

b. *not sure, makes life easier, would like more time on it*

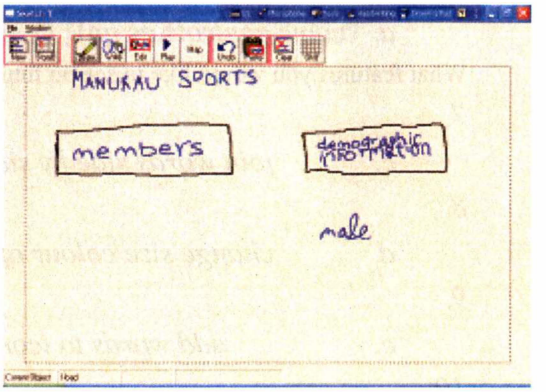
c.

**Sketches Created by Participants**

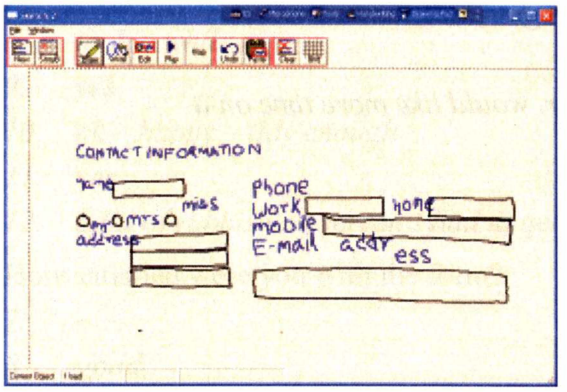
P1 storyboard



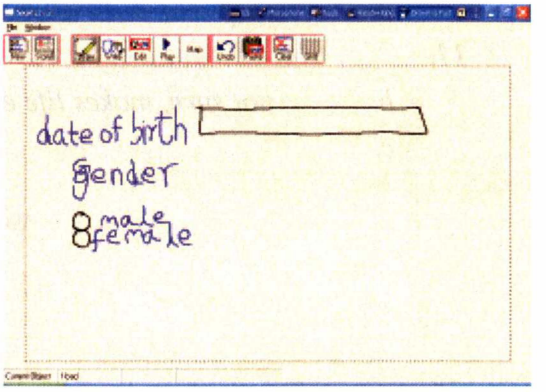
P1 Menu



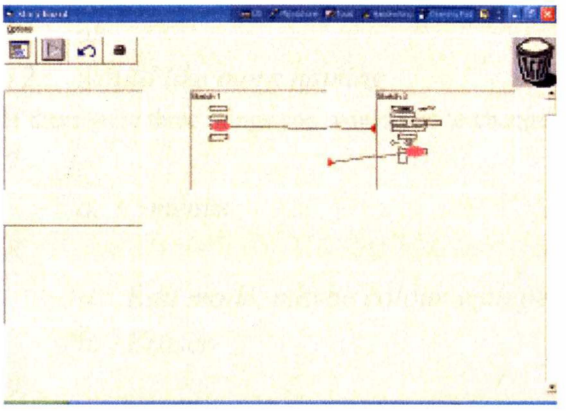

P1 Membership (a)



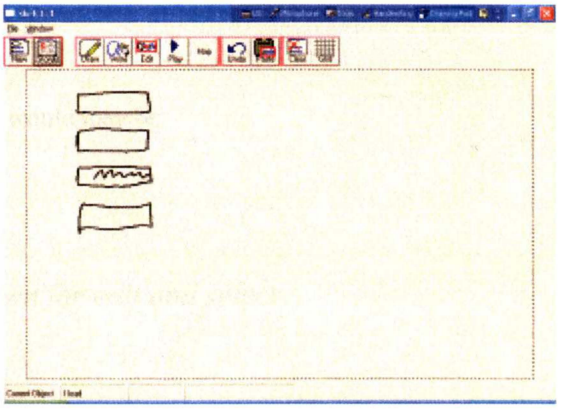

P1 Membership (b)



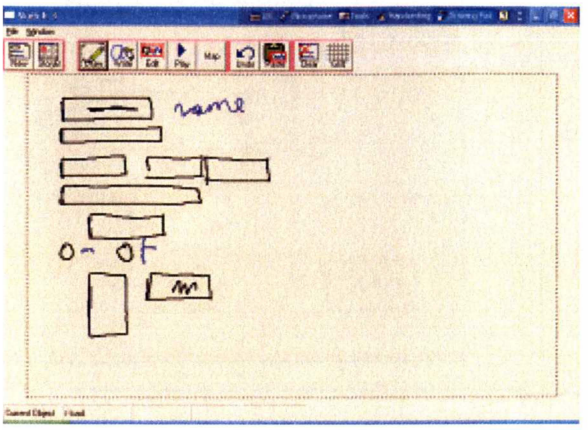

P2 Story board



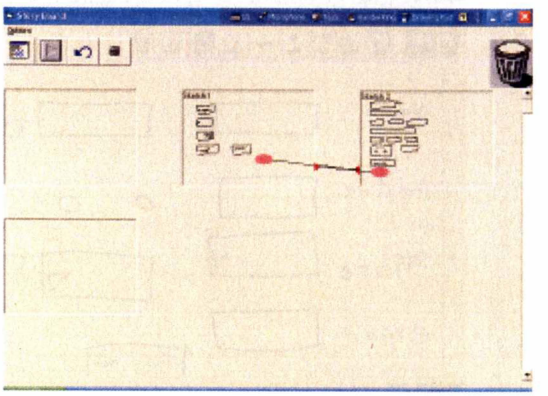
P2 Menu



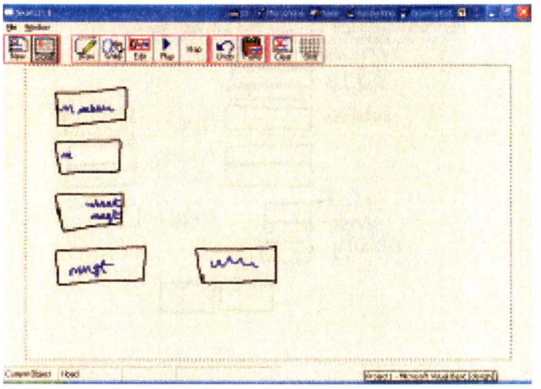
P2 Membership Details



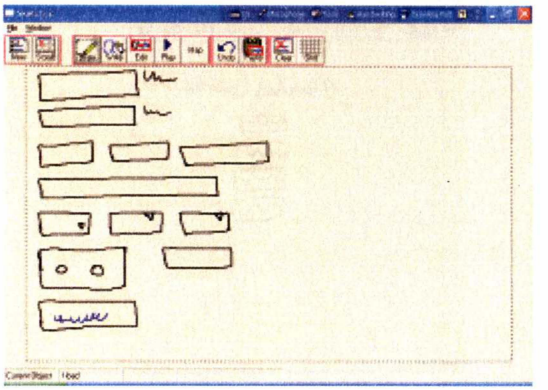

P3 Story board



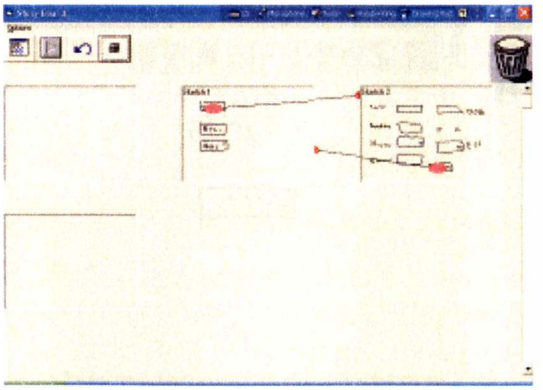
P3 Menu



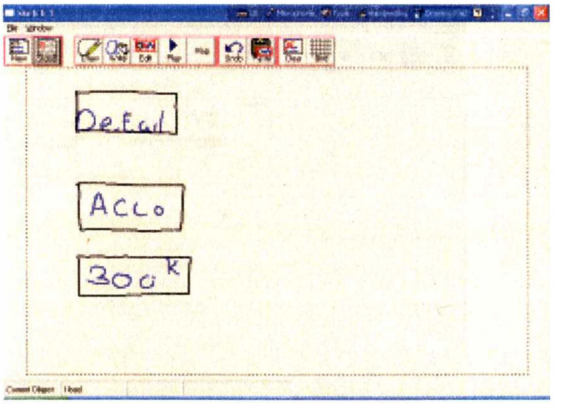
P3 Membership Details



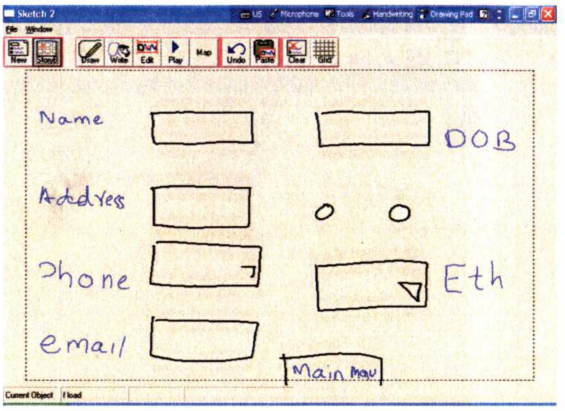
P4 Story board



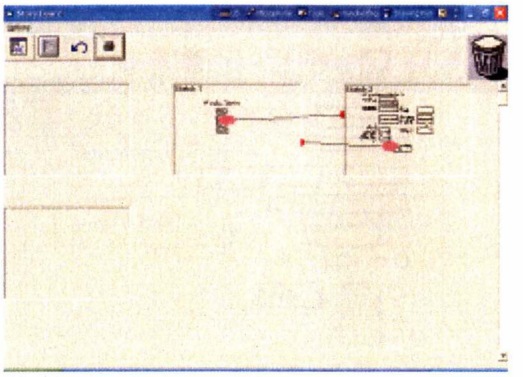
P4 Menu



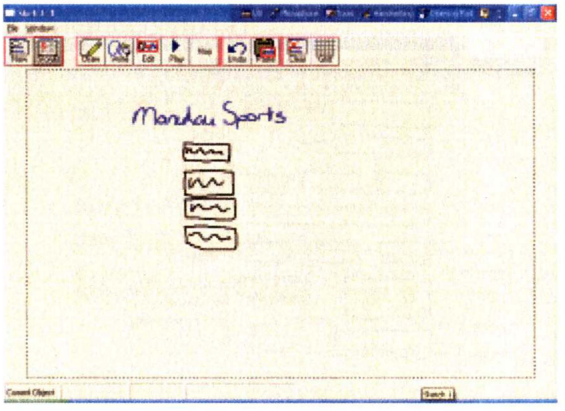
P4 Membership Details



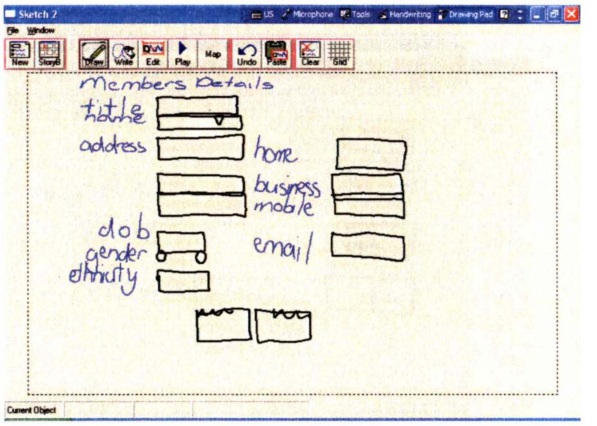

P5 Story board



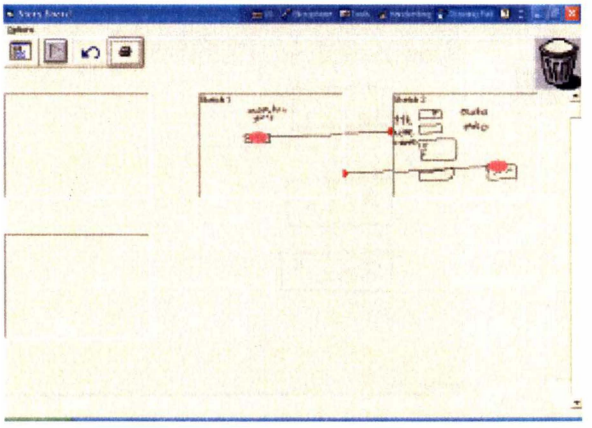
P5 Menu



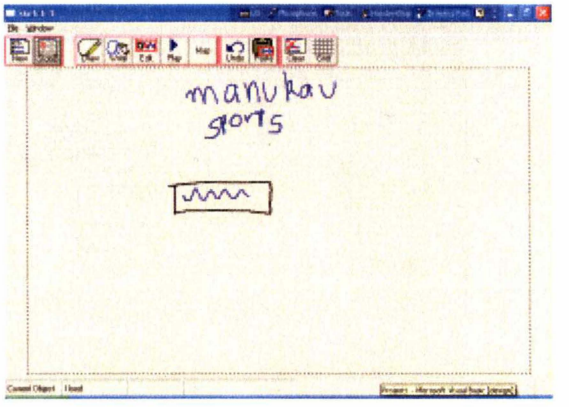

P5 Membership Details



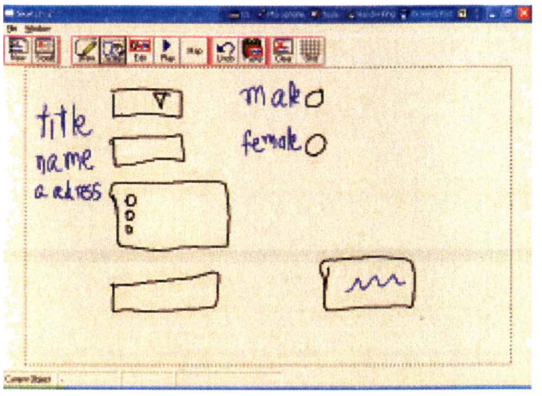

P6 Story board



P6 Menu



P6 Membership Details



# Appendix 3

# Evaluation Study

## 3.01 Sample Questionnaire

(each group had individualised questionnaires)

### W/B & Book Catalogue then FreeForm & Dog

**1. Carefully read the requirements for this form**

Design an interface for adding a new book to a catalogue. As the database will allow users to search by author, title or genre, care must be taken not to duplicate entries in these fields eg you do not want the author Tolkien recorded as both J.R.R. Tolkien and John Ronald Reuel Tolkien.

For each book you will need to record;

- title – book titles are unformatted text and may be quite long
- authors – remember a book can have more than one author, 2 authors is common, but some may have up to 6. Each author will be held separately in the database and you must be able to add or select individual authors to add to a book's author list
- isbn – international standard book number. All books have a unique isbn, it is usually about 11-12 digits broken into 4 sections, there are varying numbers of digits in each section
- year of publication – notice it is year not a date
- publisher – there is a relatively short list of popular publishers so the users should be able to choose from the list if it a publisher already in the system,
- genre – there aren't many of these
- size – height and width in centimetres
- hardback / paperback

**2. Sketch a design for the form on the whiteboard, draw a form border and all of the required controls visually indicating the control type by the shape and size of the objects, static labels and captions are indicated by writing the word on the design**

**3. Check your form with the scenarios below. Check that you have a control for each data item and that each control is of the appropriate type and size**

Scenario 1

Register a new book by a new author

Title – "My Arm", Author – Rob Finnerty, isbn 0 454 543453 0, year of publication 2002, publisher – Southland press, genre – thriller, size h 12.4cm, w 9.2cm, paperback

Scenario 2

Register a new edition of a Visual Basic Textbook.
Title – "VB.Net", Authors – Karen Programmer and Susan Superteacher, isbn 03 87 199888 10, year of publication 2002, publisher – Prentice Hall, genre – textbook, size h 31cm w 25cm, hardback

## 4. Correct any design problems you have uncovered in your checking

## 5. Create the vb form to the point where you are ready to start coding

## Problem B Dog License Application

## 1. Carefully read the requirements for this form

Design an interface to record dog registration details.
For each dog you will need to record

- Owner's
  - Name
  - Address
  - Date of Birth (this will be deployed as a calendar control but for design purposes put a textbox on the form)
  - Phone
- Dog's
  - Location of dog (if different from owner's address)
  - Name
  - Colour
  - Breed
  - Sex
  - Age
  - special purpose dog (eg guide dog) – these dogs pay no fees
  - fee payable – this will be calculated by the program depending on the information provided

## 2. Sketch a design in the FreeForm environment following the instructions you have been given. Focus on the design, while remembering the fundamentals required for recognition.

## 3. Use the FreeForm run mode to check your form with the scenarios provided. Check that you have a control for each data item and that each control is of the appropriate type and size

Scenario 1

Register a family mutt

Owners – Stuart Dogman, 24 Westbury Rd, Hamilton, DOB 24-9-1965, Phone 7-987 343.

Dog – location same as owners, name – Alfred, colour – black, breed – mixed, male, 3 years old

Scenario 2

Register a guide dog

Owners – Guide Dog Services, 24 Railway Place, Manurewa, Phone 9 298 3453.

Dog – 6 Firth Rd, Hamilton, name – Joni, colour – golden, breed – Labrador, female, 5 years old

4. **Correct any design problems you have uncovered in your checking**

5. **At this point the researcher will show you how to do the recognition and conversion to a VB form. Convert the sketch to a vb form, get the form to a point where you are ready to start coding.**

# Questionnaire

*Group Number ___1.1___* *Individual _____*

**Complete only This Section before the session**

| | | | | | |
|---|---|---|---|---|---|
| I use a whiteboard for planning task | Always | Usually | Sometimes | Rarely | Never |
| I hand sketch interface designs before I create them in the programming IDE | Always | Usually | Sometimes | Rarely | Never |
| I think sketching an interface design is necessary when planning a program | Strongly Agree | Agree | Neutral | Disagree | Strongly Disagree |

---

**The following sections will be completed in two parts**

**Complete this section after the first exercise**

**Problem** Book Catalogue

**Environment** whiteboard & VB

**Complete this section after the second exercise**

**Problem** Dog

**Environment**

Registration

freeform & VB

| | Strongly Agree | Agree | Neutral | Disagree | Strongly Disagree | Strongly Agree | Agree | Neutral | Disagree | Strongly Disagree |
|---|---|---|---|---|---|---|---|---|---|---|
| **General Questions** | | | | | | | | | | |
| This exercise was enjoyable | | | | | | | | | | |
| This exercise has helped me to understand the software design process | | | | | | | | | | |
| This exercise motivated me to find out more about programming | | | | | | | | | | |
| **About the problem** | | | | | | | | | | |
| I understand the problem | | | | | | | | | | |
| This environment helped with my problem understanding | | | | | | | | | | |
| I feel prepared to complete the program | | | | | | | | | | |
| **About the environment** | | | | | | | | | | |
| Creating the sketch was easy | | | | | | | | | | |
| Checking the scenarios was easy | | | | | | | | | | |
| I would like to use this method of program planning in the future | | | | | | | | | | |
| This experience has made me value sketching interface design more | | | | | | | | | | |

Given a choice I would like to do my informal design with

whiteboard ☐ FreeForm ☐

none ☐

## 3.02 Questionnaire Data

| Group Number | 1 | | 2 | | | 3 | | | 4 | | 5 | | 6 | | | 7 | | | 8 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Individual | a | b | a | b | c | a | b | c | a | b | a | b | a | b | c | a | b | c | a | c |
| I Use a Whiteboard for Planning tasks | 5 | 5 | 5 | 5 | 4 | 4 | 4 | 5 | 5 | 2 | 4 | 5 | 3 | 4 | 5 | 5 | 5 | 5 | 5 | 4 |
| I hand sketch interface designs before I create them in the programming IDE | 5 | 3 | 4 | 2 | 3 | 3 | 3 | 2 | 5 | 2 | 3 | 2 | 2 | 4 | 4 | 2 | 3 | 4 | 3 | 3 |
| I think sketching an interface design is necessary when planning a program | 2 | 3 | 1 | 2 | 2 | 1 | 2 | 2 | 2 | 2 | 3 | 2 | 2 | 3 | 3 | 1 | 2 | 2 | 2 | 2 |
| First Exercise Combinations | w/b and book | | | | | w/b and dog | | | | | FF and book | | | | | FF and dog | | | | |
| This exercise was enjoyable | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 1 | 1 | 2 | 2 | 2 | 3 | 1 |
| This exercise has helped me to understand the software design process | 2 | 3 | 1 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 1 | 2 | 2 | 1 | 2 | 2 | 2 | 2 | 2 |
| This exercise motivated me to find out more about programming | 2 | 4 | 2 | 2 | 2 | 2 | 2 | 3 | 2 | 3 | 1 | 2 | 3 | 2 | 2 | 1 | 1 | 2 | 1 | 1 |
| I understand the problem | 2 | 2 | 2 | 2 | 2 | 1 | 2 | 2 | 1 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| This environment helped with my problem understanding | 2 | 3 | 2 | 2 | 2 | 1 | 2 | 2 | 3 | 2 | 3 | 1 | 2 | 2 | 3 | 2 | 2 | 2 | 2 | 2 |
| I feel prepared to complete the program | 2 | 3 | 3 | 3 | 2 | 2 | 2 | 3 | 2 | 1 | 2 | 2 | 3 | 2 | 2 | 1 | 2 | 2 | 2 | 2 |
| Creating the sketch was easy | 2 | 1 | 4 | 2 | 3 | 2 | 2 | 2 | 2 | 1 | 3 | 2 | 4 | 3 | 2 | 2 | 2 | 3 | 3 | 2 |
| Checking the scenarios was easy | 2 | 3 | 4 | 2 | 2 | 2 | 2 | 3 | 3 | 2 | 2 | 1 | 4 | 2 | 2 | 3 | 2 | 3 | 2 | 2 |
| I would like to use this method of program planning in the future | 2 | 2 | 2 | 3 | 2 | 2 | 2 | 2 | 3 | 2 | 2 | 1 | 3 | 2 | 1 | 2 | 2 | 3 | 3 | 1 |
| This experience has made me value sketching interface design more | 1 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 3 | 2 | 2 | 2 | 1 | 1 | 2 | 1 | 2 | 1 | 1 |
| Second Exercise Combinations | FF and dog | | | | | FF and book | | | | | w/b and dog | | | | | w/b and book | | | | |
| This exercise was enjoyable | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 3 | 3 | 2 | 2 | 2 | 2 | 2 | 2 | 3 | 3 |
| This exercise has helped me to understand the software design process | 1 | 2 | 1 | 2 | 2 | 1 | 1 | 2 | 2 | 3 | 3 | 2 | 2 | 3 | 2 | 2 | 2 | 2 | 2 | 3 |
| This exercise motivated me to find out more about programming | 1 | 2 | 1 | 1 | 1 | 1 | 1 | 2 | 2 | 2 | 4 | 2 | 2 | 3 | 4 | 1 | 1 | 3 | 3 | 3 |
| I understand the problem | 1 | 1 | 2 | 2 | 2 | 1 | 1 | 1 | 1 | 1 | 2 | 2 | 2 | 2 | 2 | 2 | 1 | 2 | 2 | 2 |
| This environment helped with my problem understanding | 1 | 1 | 2 | 2 | 2 | 1 | 1 | 1 | 2 | 2 | 3 | 2 | 2 | 3 | 3 | 2 | 1 | 2 | 2 | 3 |
| I feel prepared to complete the program | 1 | 1 | 3 | 2 | 2 | 1 | 1 | 2 | 1 | 1 | 3 | 2 | 2 | 2 | 2 | 2 | 1 | 2 | 2 | 2 |
| Creating the sketch was easy | 1 | 2 | 4 | 1 | 1 | 1 | 1 | 2 | 2 | 1 | 2 | 2 | 2 | 2 | 2 | 3 | 1 | 1 | 2 | 2 |
| Checking the scenarios was easy | 1 | 1 | 3 | 1 | 2 | 1 | 1 | 2 | 2 | 1 | 3 | 3 | 2 | 3 | 3 | 3 | 2 | 1 | 3 | 3 |
| I would like to use this method of program planning in the future | 1 | 1 | 2 | 1 | 1 | 1 | 1 | 2 | 2 | 2 | 3 | 2 | 2 | 3 | 3 | 2 | 2 | 2 | 3 | 3 |
| This experience has made me value sketching interface design more | 1 | 1 | 2 | 1 | 1 | 1 | 1 | 1 | 2 | 1 | 2 | 2 | 2 | 2 | 2 | 2 | 1 | 2 | 2 | 3 |
| Final Question | | | | | | | | | | | | | | | | | | | | |
| Given a choice I would like to do my informal design with | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 3 | 2 | 2 | 2 | 3 | 2 | 1 | 2 |

```
Complete analysis of question 'I understand the problem'
***** Analysis of variance *****

Variate: I_understand
Source of variation      d.f.       s.s.       m.s.     v.r.  F pr.
 gpID stratum
session.environ             1    1.60000    1.60000     9.37  0.038
session.prob                1    0.90000    0.90000     5.27  0.083
environ.prob                1    0.40000    0.40000     2.34  0.201
Residual                    4    0.68333    0.17083
gpID.session stratum
session                     1    0.90000    0.90000     5.27  0.083
environ                     1    0.40000    0.40000     2.34  0.201
prob                        1    0.10000    0.10000     0.59  0.487
session.environ.prob        1    0.00000    0.00000     0.00  1.000
Residual                    4    0.68333    0.17083     2.24
gpID.ID stratum            12    0.91667    0.07639     1.00
gpID.session.ID stratum    12    0.91667    0.07639
Total                      39    7.50000
* MESSAGE: the following units have large residuals.
gpID 1.1         -0.300   s.e. 0.146
gpID 1.1   session 1st          0.300   s.e. 0.146
gpID 1.1   session 2nd         -0.300   s.e. 0.146
gpID 2.1   ID 6           -0.333   s.e. 0.151
gpID 4.1   ID 17          -0.333   s.e. 0.151
gpID 2.1   session 1st   ID 6           -0.333   s.e. 0.151
gpID 2.1   session 2nd   ID 6            0.333   s.e. 0.151
gpID 4.1   session 1st   ID 17           0.333   s.e. 0.151
gpID 4.1   session 2nd   ID 17          -0.333   s.e. 0.151
***** Tables of means *****
Variate: I_understand
Grand mean  1.750
 session      1st      2nd
             1.900    1.600
  environ freeform     whbd
             1.650    1.850
     prob     book      dog
             1.700    1.800
  session  environ freeform     whbd
      1st             2.000    1.800
      2nd             1.300    1.900
  session     prob     book      dog
      1st             2.000    1.800
      2nd             1.400    1.800
  environ     prob     book      dog
 freeform             1.500    1.800
     whbd             1.900    1.800
           environ freeform             whbd
  session     prob     book      dog     book      dog
      1st             2.000    2.000    2.000    1.600
      2nd             1.000    1.600    1.800    2.000
*** Standard errors of differences of means ***
Table              session     environ        prob     session
                                                       environ
rep.                    20          20          20          10
s.e.d.              0.1307      0.1307      0.1307      0.1848
d.f.                     4           4           4           4
Except when comparing means with the same level(s) of
 session                                                 0.1848
 d.f.                                                         8
 environ                                                 0.1848
```

```
 d.f.                                                          8
Table              session      environ      session
                      prob         prob      environ
                                                 prob
rep.                    10           10            5
s.e.d.              0.1848       0.1848       0.2614
d.f.                     4            4            4
Except when comparing means with the same level(s) of
 session            0.1848                    0.2614
 d.f.                    8                         8
 environ                          0.1848       0.2614
 d.f.                                8            8
 prob               0.1848       0.1848       0.2614
 d.f.                    8            8            8
 session.environ                              0.2614
 d.f.                                            8
 session.prob                                 0.2614
 d.f.                                            8
 environ.prob                                 0.2614
 d.f.                                            8
```

Analysis of change in opinion between initial question "I think sketching an interface design is necessary when planning a program" and session question "This experience has made me value sketching interface design more"

| Mean Change | | | | |
|---|---|---|---|---|
| Freeform | Whiteboard | Degrees of freedom | Standard error difference | P value |
| 0.70 | 0.05 | 4 | 0.20 | 0.032 |

# 3.03 Marks for Designs by Independent Expert

| | WhiteBoard | | | Freeform | | |
|---|---|---|---|---|---|---|
| group | 1st or 2nd exercise | problem | score | 1st or 2nd exercise | problem | score |
| 1.1 | 1st | book | 6 | 2nd | dog | 8 |
| 1.2 | 1st | book | 8 | 2nd | dog | 6 |
| 2.1 | 1st | dog | 6 | 2nd | book | 7 |
| 2.2 | 1st | dog | 5 | 2nd | book | 6 |
| 3.1 | 2nd | dog | 7 | 1st | book | 8 |
| 3.2 | 2nd | dog | 7 | 1st | book | 8 |
| 4.1 | 2nd | book | 6 | 1st | dog | 6 |
| 4.2 | 2nd | book | 6 | 1st | dog | 5 |
| | | | 6.375 | | | 6.75 |

| Paired T-test | | |
|---|---|---|
| Mean | 6.75 | 6.37 |
| St Dev | 1.16 | 0.91 |
| St E Mean | 0.41 | 0.46 |

95% confidence interval for mean difference -0.715 – 1.465
t test of mean difference = 0 (vs not = 0): t value = 0.81 p value = 0.44

# Changes during checking

| | Whiteboard | | Freeform | |
|---|---|---|---|---|
| group | exercise | changes | exercise | changes |
| 1.1 | 1st | 1 | 2nd | 5 |
| 1.2 | 1st | 2 | 2nd | 2 |
| 2.1 | 1st | 2 | 2nd | 6 |
| 2.2 | 1st | 0 | 2nd | 0 |
| 3.1 | 2nd | 0 | 1st | 5 |
| 3.2 | 2nd | 0 | 1st | 4 |
| 4.1 | 2nd | 2 | 1st | 4 |
| 4.2 | 2nd | 2 | 1st | 2 |
| Paired T-test | | | | |
| Mean | | 1.13 | | 3.50 |
| St Dev | | 0.99 | | 2.00 |
| St E Mean | | 0.35 | | 0.70 |

95% confidence interval for mean difference 0.59 – 4.16
t test of mean difference = 0 (vs not = 0): t value = 3.15 p value = 0.016

# 3.05 Supplementary Study

- **Problem definitions and scenarios**

**Problem Book Catalogue Form**

## 1. Carefully read the requirements for this form

This design is an interface for adding a new book to a catalogue. As the database will allow users to search by author, title or genre, care must be taken not to duplicate entries in these fields eg you do not want the author Tolkien recorded as both J.R.R. Tolkien and John Ronald Reuel Tolkien.

For each book you will need to record;

- title
- authors
- isbn
- year of publication – notice it is year not a date
- publisher
- genre
- size
- hardback / paperback

## 2. Check the form provided the scenarios below.

Scenario 1

Register a new book by a new author

Title – "My Arm", Author – Rob Finnerty, isbn 0 454 543453 0, year of publication 2002, publisher – Southland press, genre – thriller, size h 12.4cm, w 9.2cm, paperback

Scenario 2

Register a new edition of a Visual Basic Textbook.

Title – "VB.Net", Authors – Karen Programmer and Susan Superteacher, isbn 03 87 199888 10, year of publication 2002, publisher – Prentice Hall, genre – textbook, size h 31cm w 25cm, hardback

## 3. Correct any design problems you have uncovered in your checking

**Problem Credit Card Application Form**

## 1. Carefully read the requirements for this form

This design is for an interface to record credit card application details.

It will need to record the applicants

- Name
- Address
- Phone Numbers
- Gender
- Employer's name and contact details
- Whether the applicant owns their own home
- The applicants net worth
- Credit limit requested. There are standard credit limits of $500, $1000, $5000, or $10,000

## 2. Check the form with the scenarios provided.

Scenario 1

Andrew Smithers (male) of 26 Westbury Rd, Hamilton, Phone 7-987 343 or mobile 025 764987 wishes to apply for a credit card. He is a student, not currently employed, does not own his own house, has a net worth of $2,300 and would like a $500 credit limit

Scenario 2

Gillian Bridgeway (female) of 45 Railway Place, Manurewa, Phone Home 9 298 3453.or work 9 7659878 also wishes to apply for a credit card. She is a programmer earning $65000 pa, she owns her home and has a net worth of $89000. She would like a $5000 credit limit

## 3. Correct any design problems you have uncovered in your checking

## Group Performances

| Group | Exercise 1 | Changes | Exercise 2 | Changes |
|---|---|---|---|---|
| 1 | VB<br>Book | 5 | Freeform<br>Credit card | 7 |
| 2 | Freeform<br>Book | 10 | VB<br>Credit card | 9 |
| 3 | VB<br>Credit card<br>(hand drawn<br>on paper) | 10 * | Freeform<br>Book | 8 |
| 4 | Freeform<br>Credit card | 11 | VB<br>Book | 6 |
| 5 | VB<br>Book | 5 | Freeform<br>Credit Card | 8 |
| 6 | Freeform<br>Credit Card | 8 | VB<br>Book | 4 |

* this value was treated as missing as the group did not work directly on the computer but transformed the computer form to a paper sketch, changed it and transformed it back.

| Paired T-test | VB | Freeform |
|---|---|---|
| Mean | 5.80 | 8.80 |
| St Dev | 1.92 | 1.64 |
| St E Mean | 0.86 | 0.73 |

95% confidence interval for mean difference 1.03-4.96
t test of mean difference = 0 (vs not = 0): t value = 4.24 p value = 0.013